LHospital

A *Turbo* C++ project
A basic management system for a general hospital

Arpit Saxena 9151996
Anirudh Panigrahi 9151993
Sankalp Gambhir 9152014

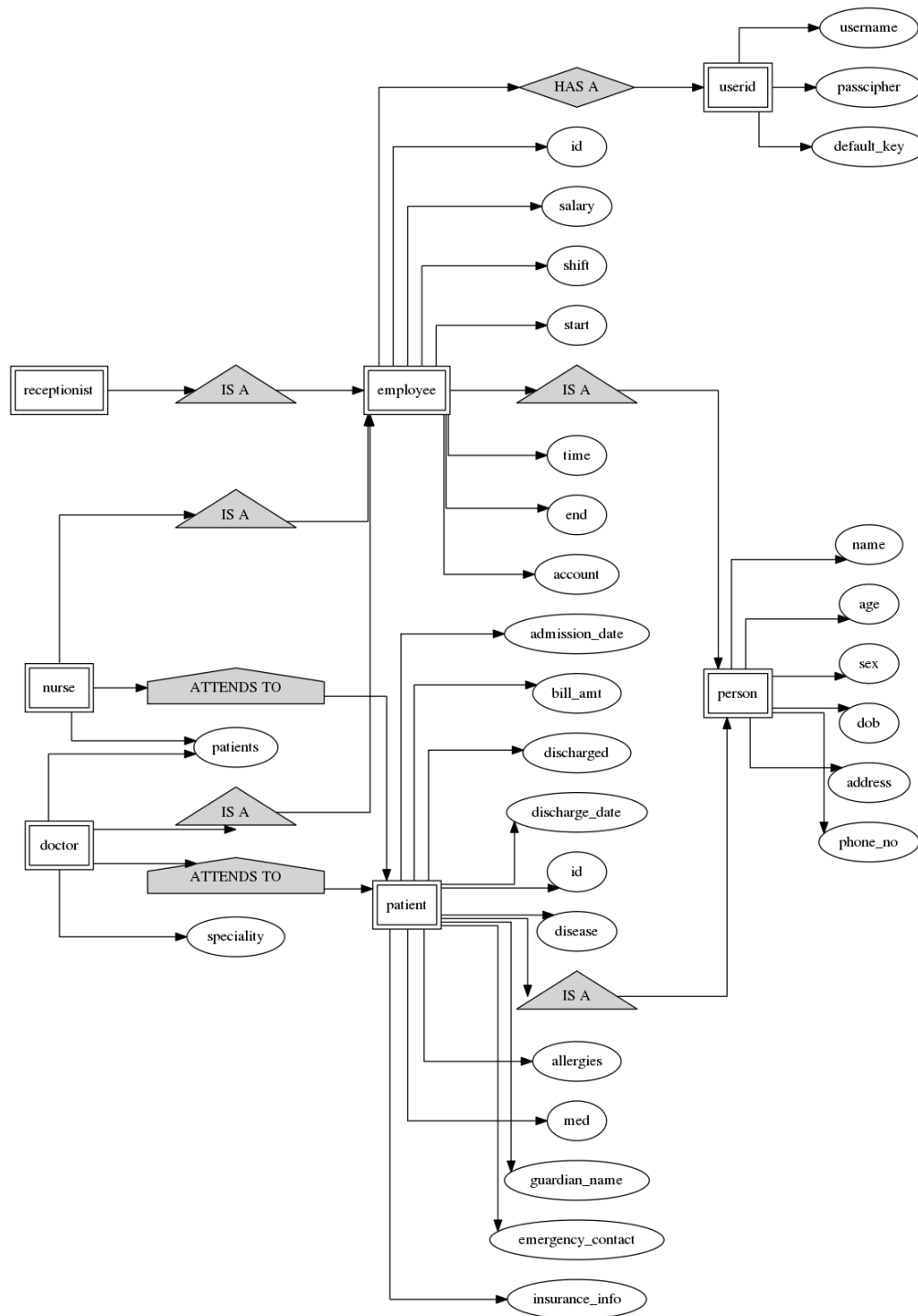This page has been intentionally left blank.

Contents

Diagrams

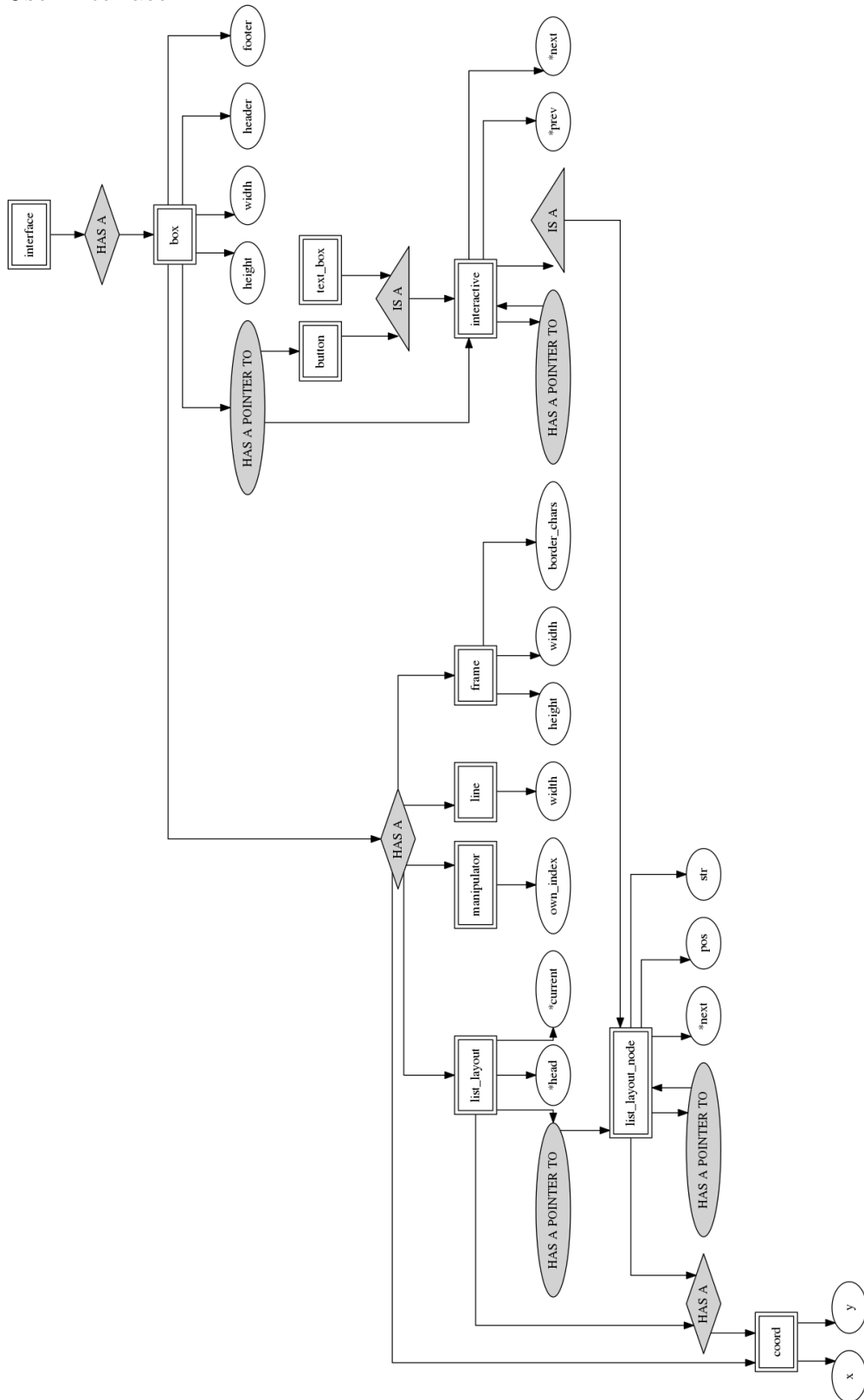## ER   diagrams
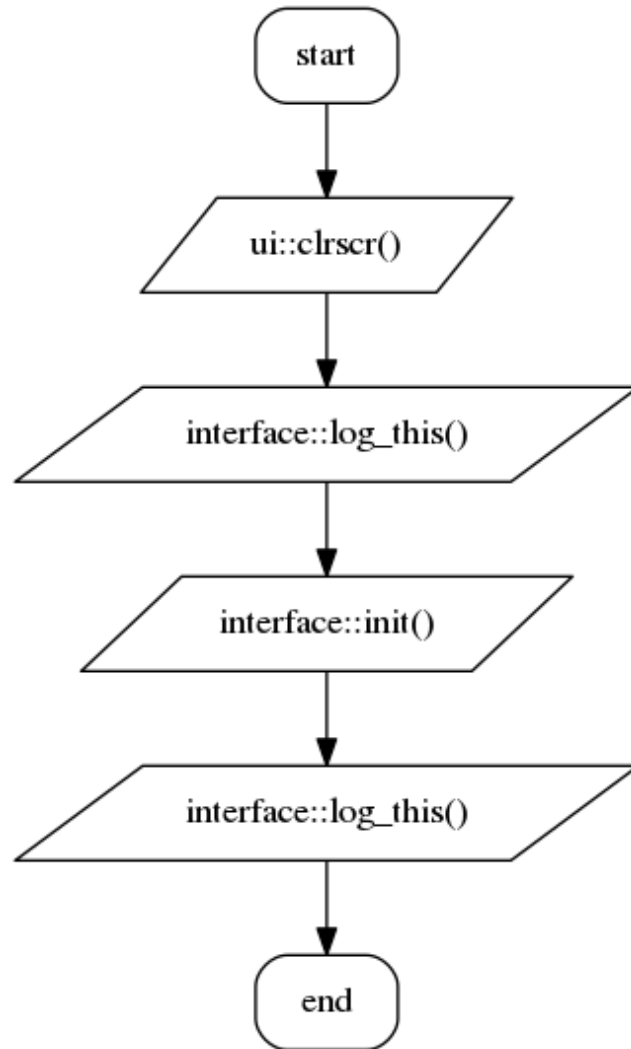
1. Hospital

2. User interface

Note: The figure included has been rotated

4

Flowchart of main()

Source Code

Header files

1. code/iface.hpp

```
1   /*!
2    \file iface.hpp
3    \brief Contains prototypes of the functions managing the interface of the
         program
4   */
5
6   #ifndef INTERFACE_HPP
7   #define INTERFACE_HPP
8
9   #include "ui/ui.hpp"
10
11  //!Class containing all the functions that make up the interface of the program
12  class interface{
13      public:
14          static void init(); //!>The main interface function of the program; this
                is the functions that runs throughout the program
15          static int login_screen();//!>Login screen interface
16          static int menu();//!>The first main menu interface(For administrator
                employee only)
17          static void patient_management(); //!>Patient management interface(for
                receptionist employees only)
18          static void employee_management(); //!>Employee management interface(for
                administrator employees only)
19          static void stock_management(); //!>Stock management interface(for
                administrator employees only)
20
21          static void employee_screen(unsigned long); //!>The interface for non—
                administrator employees
22
23          static void error(char*); //!>Prints an error message at the footer of
                interface::window
24          static void clear_error(); //!>Clears the error message at the footer of
                interface::window
25
26          static int log_this(char *); //!>Logs a message string into a file log.
                txt
27
28      protected:
29          interface(); //!>Objects of this class shouldn't be created
30          /*!
31          For creating a validation function to use in menus
32          to validate the choice input of the menu option to be accessed
33          */
34          class validate_menu
35          {
36                  static int lowest_choice, greatest_choice; //!>The lower and
                    upper limit of the choices of a menu
37                  validate_menu(); //!>Objects of this class shouldn't be created
38              public:
39                  static int input(const char *); //!>The validation function that
                    will be passed as an argument to box::operator>>()
```

6

```
40              static void set_menu_limits(int, int); //!>Setter; sets
                     lowest_choice and greatest_choice
41          };
42          /*!Creates a back_func that can be passed as an argument to box::
               setback_func()*/
43          class back_func
44          {
45              back_func(); //!>Objects of this class shouldn't be created
46          public:
47              static int backbit; //!>1, if shift + bkspc is pressed, 0
                     otherwise
48              static int set_backbit(); //!>Setter, passed as an argument to
                     box::setback_func()
49          };
50          static box window; //!>The main outer window box
51  };
52
53  //!Class containing all the functions that make up the interface of Employee
        management
54  class emp_mgmt : public interface
55  {
56      public:
57          static void view_emp(); //!>Interface of View Employee
58          static int view_emp(unsigned long); //!>Creates the interface that shows
                the details of an employee with a particular ID
59          static void add_emp(); //!>Interface of Add Employee
60          static void remove_emp(); //!>Interface of Remove Employee
61          static void edit_emp(); //!>Interface of Edit Employee
62          static void pay_emp(); //!>Interface of Pay Employee
63          static void pay_all(); //!>Interface of Pay All Employees
64      private:
65          emp_mgmt(); //!>Objects of this class shouldn't be created
66  };
67
68  #endif /* INTERFACE_HPP */
```

2. code/EMP.HPP

```
1  /*!
2   \file EMP.HPP
3   \brief Contains the definitions of the employee class and its derivatives
4  */
5
6  #ifndef EMP
7  #define EMP
8
9  #include "base.hpp"
10
11 enum emp_type {INVALID, OTHERS, DOCTOR, NURSE, RECEPTIONIST};
12 //!>Identifiers for indication of different types of employees
13
14 //!Class storing details of employees of the hospital
15 class employee : public person{
16     int generate_id(); //!>Generates ID of the employee
17     static int generate_id_status; //!>0 if the last id generation was
            unsuccessful
18     /*!>Basically ensures that id generation is stopped when an error occurs
            in id generation,
```

```
19          otherwise, the files(max_id.dat, id_list.dat) might start storing
               meaningless data, which
20          will affect future id generation*/
21
22      public:
23          employee(str, int, Date, address, phone, unsigned long, Time, Time, str =
               "", str = ""); //!>Explicit constructor
24          /*!>for all those with user accounts(doctors, nurses, receptionists),
25          last 2 arguments are to be provided as well*/
26          employee(); //!>Default constructor
27
28          //!@{Getters
29          int get_age(); //!>Overridden function
30          /*!>Updates the age of the employee and writes the employee object back
               to file before returning age*/
31          unsigned long get_salary();
32          Time get_shift(int inp1); /*!>\param inp1 times_of type variable that
               indicates starting or ending shift time*/
33          unsigned long get_id();
34          static int get_generate_id_status();
35          transaction * get_last_5_transactions(); //!>Gets the last 5 records
               present in the file TRANS.DAT of the employee's folder
36          //!}@
37
38          //!@{Setters
39          void set_salary(unsigned long);
40          void set_shift(int inp1, Time t1);/*!>\param inp1 times_of type variable
               that indicates starting or ending shift time*/
41          //!}@
42
43          userid account; //!>Facilitates login mechanism of the employee
44      protected:
45          unsigned long id; //!>ID of the employee
46          unsigned long salary; //!>Salary of the employee
47          Time shift_start; //!>Starting shift time of the employee
48          Time shift_end; //!>Ending shift time of the employee
49  };
50
51  //!Class storing details of doctors of the hospital
52  class doctor : public employee{
53      public:
54          doctor(str, int, Date, address, phone, unsigned long, Time, Time, int,
               int, str, str); //!>Explicit constructor
55          doctor(); //!>Default constructor
56
57          //!@{Getters
58          int * get_speciality();
59          long * get_patients();
60          //!}@
61
62          //!@{Setters
63          void set_speciality(int *);
64          void set_patients(long *);
65          //!}@
66
67      private:
68          int speciality[2]; //!>Doctor's specialization
69          long patients[10]; //!>Patients currently under care, can take only 10 at
               once
```

```
70   };
71
72   //!Class storing details of nurses of the hospital
73   class nurse : public employee{
74       public:
75           nurse(str, int, Date, address, phone, unsigned long, Time, Time, str, str
                 ); //!>Explicit constructor
76           nurse(); //!>Default constructor
77           long * get_patients(); //!>Getter
78
79           void set_patients(long *); //!>Setter
80       private:
81           long patients[5]; //!>Patients currently under care, can take only 5 at
                 once
82   };
83
84   //!Class storing details of receptionists of the hospital
85   class receptionist : public employee
86   {
87       public:
88           receptionist(str, int, Date, address, phone, unsigned long, Time, Time,
                 str, str); //!>Explicit constructor
89           receptionist(); //!>Default constructor
90   };
91
92   //!Class that generates objects storing the employee type corresponding to each
         id
93   /*!
94   This class is used to generate objects storing the employee type corresponding to
          each id,
95   and then to store these objects to a file EMPLOYEE/ID_LIST.DAT(The ctor itself
         does all this)
96   This class is used to get the employee type of an employee having a particular id
97   */
98   class id_to_emp
99   {
100          unsigned long id;//!>ID of employee
101          int employee_type;//!>Type of employee
102      public:
103          int status; //!>True whenever the constructor runs successfully and
                 succeeds in storing the object to id_list.dat
104          id_to_emp(unsigned long, int); //!>Explicit constructor
105          id_to_emp(); //!>Default constructor
106          static int convert(unsigned long); //!>Converts id to employee type
107  };
108
109  #endif
```

3. code/BASE.HPP

```
1   /*!
2    \file BASE.HPP
3    \brief Contains the declarations of the basic structs, classes, typedefs
4           and enums to be used in the whole program
5   */
6
7   #ifndef BASE
8   #define BASE
9
```

```cpp
10   #include "ui/ui.hpp"
11   #include <fstream.h>
12   #include <string.h>
13   #include <dir.h>
14   #include <stdio.h>
15   #include <math.h>
16   #include <string.h>
17   #include <time.h>
18   #include <stdlib.h>          //for random() and randomize()
19
20   typedef char str[80]; //!>typedef for general strings
21   typedef char phone[11]; //!>typedef for strings storing phone numbers
22
23   enum sex {MALE, FEMALE, TRANS}; //!>Identifiers for different sexes
24   enum date_type {DAY, MONTH, YEAR}; //!>Identifiers for different parts of a date
25   enum time_type {HOUR, MINUTE, SECOND}; //!>Identifiers for different parts of a
        time
26   enum body_parts {BRAIN, HEART, SKIN,
27       LUNG, BONE, EYE,
28       THROAT, TEETH, STOMACH,
29       BLOOD, GUT, GEN}; //!>Identifiers for different parts of the human body
30   /*!>used for recording specialities of doctors(GEN for general problems)*/
31
32   enum address_parts {HOUSE_NO, STREET, CITY, DISTRICT, STATE}; //!>Identifiers for
        different parts of an address
33   enum times_of {START, END}; //!>Identifiers indicating start or end of something
34   /*!>(used in get_shift() and set_shift() to get or set starting or ending shift
        time)*/
35
36   struct Time{
37       unsigned int hour;
38       unsigned int minute;
39       unsigned int second;
40
41       Time();
42       Time(unsigned h, unsigned m, unsigned s);
43   };//!>Structure facilitating implementation of a time variable
44
45   struct Date{
46       unsigned int day;
47       unsigned int month;
48       unsigned int year;
49
50       Date();
51       Date(unsigned d, unsigned m, unsigned y);
52   };//!>Structure facilitating implementation of a date variable
53
54   class system
55   {
56       private:
57           system();
58       public:
59           static Date get_date();
60           static Time get_time();
61   };//!>Contains prototypes of functions that return the system date and time
62
63   struct address{
64       str house_no;
65       str street;
```

```cpp
 66        str city;
 67        str district;
 68        str state;
 69
 70        address(const char * = "", const char * = "", const char * = "", const char *
              = "", const char * = "");
 71  };//!>Structure facilitating implementation of an address variable
 72
 73  struct disease{
 74        str name;
 75        int type;              //refers to body part affected (LUNG, HEART, etc)
 76        str symptoms[4];      //symptoms reported by patient
 77  };//!>Structure facilitating implementation a variable storing details of a
         disease
 78
 79  struct insurance{
 80        str provider;
 81        unsigned long amount;
 82        Date expiry;
 83  };//!>Structure facilitating implementation a variable storing insurance details
         of any person
 84
 85  struct medicine{
 86        int code;
 87        float price;
 88        str name;
 89        float dosage;
 90        long stock;
 91  };//!>Structure facilitating implementation a variable storing details of a
         medicine
 92
 93  struct transaction{
 94        float amount;
 95        str reason;
 96        Date _date;
 97        Time _time;
 98        transaction(float, Date = Date(), Time = Time(), char* = "NA");
 99        transaction();
100  };//!>Structure facilitating implementation a variable storing details of a
         transaction
101
102  struct procedure{
103        str name;
104        float cost;
105  };//!>Structure facilitating implementation a variable storing details of a
         medical procedure
106
107  //!Class storing all common data members of a person
108  /*!
109  Parent class to all the persons that this program handles, i.e patients,
110  and all types of employees.
111  */
112  class person{
113        public:
114            person(str, int, Date, address, phone); //!>Explicit constructor
115            person(); //!>Default constructor
116
117            //!@{Getters
118            char* get_name();
```

11

```cpp
119            int get_age();
120            int get_sex();
121            Date get_dob();
122            address get_address();
123            char* get_phone();
124            //!}@
125
126            //!@{Setters
127            void set_name(char*);
128            void set_sex(int);
129            void set_dob(Date, Date = system::get_date());
130            void set_address(address);
131            void set_phone(char*);
132            //!}@
133
134        protected:
135            str name; //!>Name of the person
136            unsigned age; //!>Age of the person
137            unsigned sex; //!>Sex of the person
138            Date dob; //!>Date of birth of the person
139            address adr; //!>Address of the person
140            phone phone_no; //!>Phone number of the person
141
142        private:
143            void calc_age(Date d = system::get_date()); //!>Calculates age of the
                      person using dob
144            /*!>\param d The date with respect to which age is to be calculated(
                      default value is set to be the system date)*/
145   };
146
147   //!Class managing login features of the program
148   /*!
149   This class stores a username and a password in encrypted form, besides
150   the inplementation data. This class uses a vigenere cipher to encrypt the
151   password and store it.
152   */
153   class userid
154   {
155            str username; //!>Username of the login account
156            str passcipher; //!>Encrypted password
157            str default_key;//!>Key for making the vigenere cipher
158            void makecipher(char *); //!>Makes the vigenere cipher
159            void set_key(char *); //!>Sets default_key to a random string
160            char * decipher(); //!>deciphers the cipher 'passcipher'
161
162        public:
163            userid(char *, char *); //!>Explicit constructor
164            userid(); //!>Default constructor
165            char * get_username(); //!>Getter
166            void set_username(char *); //!>Setter
167            int login(char *); //!>\return 1 if the string input in the function is
                      the password, 0 otherwise
168   };
169
170   //!Defines << operator overloads to facilitate printing of some stuff
171   class enum_to_str
172   {
173            enum_to_str();
174        public:
```

```
175        friend box & operator<<(box &output, sex s);          //!>converts sex
               enumeration constant into a string and prints it to a box
176        friend box & operator<<(box &output, body_parts b); //!>converts
               body_parts enumeration constant into a string and prints it to a box
177        friend box & operator<<(box &output, Time & t);       //!>converts Time
               variable into a string and prints it to a box
178        friend box & operator<<(box &output, Date & d);       //!>converts Date
               variable into a string and prints it to a box
179        friend box & operator<<(box &output, address & a);  //!>converts address
               variable into a string and prints it to a box
180  };
181
182  #endif
```

4. code/PATIENT.HPP

```
1   /*!
2    \file PATIENT.HPP
3    \brief Contains the patient class definition
4   */
5
6   #ifndef PATIENT
7   #define PATIENT
8
9   #include "base.hpp"
10
11  //!Base patient class
12  class patient : public person
13  {
14      protected:
15
16          long id; //!>Unique identification number of patient
17          disease dis; //!>patient's afflictions
18          str allergies[2]; //!>patient's known allergies
19          int med[50][2]; //!>patient's purchased medicines & quantities
20          str guardian_name; //!>Patient's guardian/caretaker
21          str emergency_contact; //!>Patient's emergency contact's name
22          phone emer_contact_no; //!>Emergency contact phone number
23          insurance insur_info; //!>Insurance struct storing patient's insurance
               info
24          Date admission_date; //!>Date patient was admitted
25          unsigned long bill_amt; //!>Net amount to be billed to patient
26          int discharged; //!>Binary; = 0 Admitted | = 1 Discharged
27          Date discharge_date; //!>Date patient was discharged. NULL if still
               admitted
28
29      public:
30
31          patient(str, int, Date, address, phone, disease, str, str, phone,
               insurance, Date = system::get_date()); //!>Explicit constructor
32          //!>If no date is provided, it is assumed that patient was admitted on
               the current system date
33          patient(); //!>Default constructor
34
35          //!@{Getters
36
37          //!Returns unique ID of patient
38          long get_id();
39          //!Returns a disease type object of patient's affliction
```

```cpp
40          disease get_dis();
41          //!Returns patient's guardian's name
42          char* get_guardian_name();
43          //!Returns emergency contact's name
44          char* get_emergency_contact();
45          //!Returns emergency contact's phone no.
46          char* get_emer_contact_no();
47          //!Returns patient's insurance info as a struct
48          insurance get_insur_info();
49          /*!Returns admission date parameters
50                  Uses enum date_type
51                      0 — DAY
52                      1 — MONTH
53                      2 — YEAR*/
54      int get_admission_date(int);
55          //!Returns current amount billed to patient
56      unsigned long get_bill_amt();
57          /*!Returns a record of medicines purchased
58                  get_med(x, y)
59                      0 <= x < 50 Number of medicine purchase in list
60                      0 <= y <= 1 | 0 — Return medicine code
61                                  | 1 — Return medicine quantity */
62      int get_med(int, int);
63          /*!Returns discharge date parameters
64                  Uses enum date_type
65                      0 — DAY
66                      1 — MONTH
67                      2 — YEAR*/
68      int get_discharge_date(int);
69          /*!Returns a transaction element by its number from
70          the patient's list of transactions*/
71      transaction get_transaction(int);
72          //!}@
73
74          //!@{Setters
75          //!Overwrites the current stored affliction for patient
76      void set_dis(disease);
77          //!Overwrites the current stored guardian name
78      void set_guardian_name(char*);
79          //!Overwrites the current stored emergency contact name
80      void set_emergency_contact(char*);
81          //!Overwrites the current stored emeegency contant phone
82      void set_emer_contact_no(char*);
83          //!Overwrites the current stored insurance information
84      void set_insur_info(insurance);
85          //!Overwrites the current stored admission date
86      void set_admission_date(Date);
87          //!Overwrites the current billed amount
88      void set_bill_amt(unsigned long);
89          /*!Overwrites a medicine record for patient
90                  get_med(x, y, z)
91                      0 <= x < 50 Number of medicine purchase in list
92                      y — medicine code
93                      z — medicine quantity */
94      void set_med(int, int, int);
95          //!Overwrites the current stored discharge date
96      void set_discharge_date(Date);
97          //!Sets patient's discharge status to 1 / TRUE
98      void discharge();
```

```
99          //!}@
100  };
101
102  #endif
```

5. code/HOSP.HPP

```
1   /*!
2    \file HOSP.HPP
3    \brief Contains prototypes of the hospital management functions
4   */
5
6   #ifndef HOSP
7   #define HOSP
8
9   #include "base.hpp"
10  #include "patient.hpp"
11
12  //!Stores the no. of days in each month of the year(for hospital::
        get_date_difference())
13  const int monthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
14  const int stay_charge = 50;  //!> The charge per day of stay in the hospital($50
        per day)
15
16  //!Class containing all the basic hospital management functions used in the
        program
17  class hospital
18  {
19      public:
20          //!@{Hospital finances management functions
21          static float get_balance();      //!<Getter function
22
23          //!Deducts the input amount from hospital::balance
24          /*!
25          \param amt The amount to be deducted from hospital::balance
26          \param reason The reason for deduction of money
27          \param dt The date of deduction of money
28          \param tm The time of deduction of money
29          \return A transaction type variable containing details about the amount
                deduction
30          */
31          static transaction deduct_money(float amt, char* reason, Date dt, Time tm
                );
32
33          //!Adds the input amount to hospital::balance
34          /*!
35          \param amt The amount to be added to hospital::balance
36          \param reason The reason for deposit
37          \param dt The date of deposit
38          \param tm The time of deposit
39          \return A transaction type variable containing details about the deposit
40          */
41          static transaction add_money(float, char*, Date, Time);
42
43          //!Returns the last 10 transactions of the hospital
44          /*!
45          Reads the last 10 records from a file transactions.dat
46          \return An array of type transaction containing those 10 records
47          */
```

```cpp
48          static transaction* get_transaction();
49          //!}@
50
51          //!@{Patient management functions
52
53          //!Gets a patient object from file corresponding to the inputted id
54          /*!
55          Reads a patient object from a file base.dat that is present in
56          a folder with name as the id of the patient. All such folders are
57          present in a folder named PATIENT
58          \param id The id of the patient to be read
59          \return The patient object read from file
60          */
61          static patient get_patient_by_id(long id);
62
63          //!Writes a patient object to file
64          /*!
65          Makes a new folder(if it doesn't exist) in PATIENT, whose name is the
66          id of the patient object that is to be written, and writes the patient
67          object to a file BASE.DAT inside that folder
68          \param a The patient object that is to be written to file
69          */
70          static void write_patient(patient a);
71
72          //!Charges a patient for any service, treatment etc. that the patient
                availed from the hospital
73          /*!
74          Gets the patient object from file using get_patient_by_id(), and adds a
75          transaction type variable to a file TRANS.DAT that is present in the
                folder
76          of the patient(i.e the folder having name as the id of the patient)
77          \param pat_id ID of the patient
78          \param trans The transaction type variable containing details of the
                transaction that is charged to the patient
79          */
80          static void charge_patient(int pat_id, transaction trans);
81
82          //!Discharges a patient from the hospital
83          /*!
84          Calls the patient::discharge() function, sets the patient's discharge
85          date to the current system date, and writes the patient back to file
86          \param temp The object of the patient who is to be discharged
87          */
88          static void discharge_patient(patient temp);
89          static float calc_bill(int); //!>Calculates the cost of the patient's
                stay in the hospital
90          //!}@
91
92          //!@{Functions for medicine records management
93
94          //!Gets a medicine from file corresponding to an input code
95          /*!
96          Reads a medicine type variable from a file STOCK/MED.DAT that has code
                equal to the
97          code inputted to the function
98          \param inp_code The code of the required medicine
99          \return The medicine corresponding to inp_code
100         */
101         static medicine get_med_by_code(int inp_code);
```

```
102
103        //!Writes a medicine to file
104        /*!
105        Writes a medicine type variable back to the file STOCK/MED.DAT which
               stores all
106        medicines, after it has been edited, into its position in the file.
107        */
108        static void write_med(medicine);
109        //!}@
110
111        //!@{Employee management functions
112
113        //!Gets an object of an employee(or its derivative) from file
               corresponding to an input id
114        /*!
115        Reads an object from a file base.dat that is present in
116        a folder with name as the id of the employee. All such folders are
117        present in a folder named EMPLOYEE, or EMPLOYEE/DOCTOR, or
118        EMPLOYEE/NURSE, or EMPLOYEE/RECEPTIONIST
119        This function converts the id inputted to it into the
120        employee type, sets the string of the path to the folder containing
121        the employee object file, and reads the object from the file to a
122        buffer pointed to by a void pointer
123        A void pointer is used in this function to handle the different data
               types
124        i.e. employee, doctor, nurse, receptionist that can be input into the
125        function as the target parameter.
126        \param id The id of the employee to be read
127        \param target void pointer pointing to the buffer that stores the object
128        \return 1 if the function executed without errors, 0 otherwise
129        */
130        static int get_employee_by_id(unsigned long id, void * target);
131
132        //!Writes an object of type employee(or its derivative) to file
133        /*!
134        This function converts the id of the object pointed to by the pointer
135        a into the employee type, sets the string of the path to the folder that
               should
136        contain the employee object file, and then makes a new folder(if it doesn
               't exist)
137        in EMPLOYEE, or EMPLOYEE/DOCTOR, or EMPLOYEE/NURSE, or EMPLOYEE/
               RECEPTIONIST
138        (depending upon the type of the object pointed by the input void pointer)
               , whose
139        name is the id of the employee object that is to be written, and writes
               the
140        employee object to a file BASE.DAT inside that folder.
141        A void pointer is used in this function to handle the different data
               types
142        i.e. employee, doctor, nurse, receptionist that can be input into the
143        function as the parameter a.
144        \param a void pointer pointing to the object that is to be written to
               file
145        \return 1 if the function executed without errors, 0 otherwise
146        */
147        static int write_employee(void * a);
148
149        //!Pays salary to an employee having a particular id
150        /*!
```

```
151        Obtains the employee's object from file using get_employee_by_id(),
152        gets the salary of the employee, deducts the salary from hospital::
              balance
153        using hospital::deduct_money(), and then writes the transaction details
              of
154        this payment to a file TRANS.DAT present in the folder of the employee (i
              .e
155        the folder having name as the id of the employee), that is present in
              folder
156        EMPLOYEE, or EMPLOYEE/DOCTOR, or EMPLOYEE/NURSE, or EMPLOYEE/RECEPTIONIST
              ,
157        depending on the employee type.
158        \param id ID of the employee to whom salary is to be paid
159        \param d1 Date of payment of salary
160        \param t1 Time of payment of salary
161        \return 1 if the function executed without errors, 0 otherwise
162        */
163        static int pay_salary(unsigned long id, Date d1, Time t1);
164
165        //!Pays salary to all employees whose files are on the disk
166        /*!
167        Loops the execution of pay_salary(); the maximum no. of times the loop
              should
168        run is determined by max_id, an unsigned long variable stored in a file
169        EMPLOYEE/MAX_ID.DAT
170        \return 1 if the function executed without errors, 0 otherwise
171        */
172        static int pay_all_salaries();
173        //!}@
174
175        //!@{Internal implementation functions
176        static int get_date_difference(Date, Date); //!>Calculates the no. of
              days between 2 dates
177        static int count_leap_years(Date); //!>Calculates the no. of leap years
              between a certain date and the year 0 AD
178        static int date_validity(const char * inp_date); //!>\return
              date_validity(str_to_date(inp_date))
179        static int date_validity(Date); //!>\return 1 if the date supplied to the
               function is a valid date, 0 otherwise
180        static int time_validity(const char * inp_time); //!>\return
              time_validity(str_to_time(inp_time))
181        static int time_validity(Time); //!>\return 1 if the time supplied to the
               function is a valid time, 0 otherwise
182        static Date str_to_date(const char *); //!>converts a string to a Date
              type variable
183        static Time str_to_time(const char *); //!>converts a string to a Time
              type variable
184        static int str_to_sex(char *); //!>converts a string to an enum sex type
              variable
185        //!}@
186
187   private:
188        hospital();//!>Objects of this class shouldn't be created
189
190        //! Reads an object from a file on disk and stores it in a buffer
191        /*!
192        A specific implementation of the fstream::read() function for the
              hospital's
193        purposes.
```

```
194          \param ID ID of the object that is to be read(for error logging purposes
                only)
195          \param dest The path string to the file from which the object is to be
                read
196          \param size Size in bytes of the object that is to be read
197          \param temp Pointer to the buffer at which the read object is to be
                stored
198          */
199          static int read_from(unsigned long ID, char * dest, int size, char *temp)
                ;
200          static double balance; //!>Current balance of the hospital
201  };
202
203  #endif
```

6. code/UI/test.hpp

```
1   #ifndef TEST_HPP
2   #define TEST_HPP
3
4   void test_weird_error();
5
6   int back_func();
7   void test_back();
8
9   void test_all();
10  void test_listlayout();
11  void test_textbox();
12  void test_frame();
13
14  #endif /* TEST_HPP */
```

7. code/UI/ui.hpp

```
1   /*!
2    \file ui.hpp
3    \brief Contains prototypes of UI functions
4   */
5
6   #ifndef UI_HPP
7   #define UI_HPP
8
9   #include <conio.h>
10  #include <stdarg.h>
11  #include <string.h>
12  #include <stdio.h>
13  #include <iostream.h>
14  #include <ctype.h>
15  #include <stdlib.h>
16  #include <limits.h>
17  #include <errno.h>
18  #include <new.h>
19  #include <process.h>
20
21  //! Validator function that's used for validating user input
22  typedef int (*validator_f)(const char *);
23
24  //! For running ui::init() before main (initialising basic stuff)
```

```
25  class init_lib_ui
26  {
27      static int counter; //!< Ensures ui::init() is called only once
28      public:
29          init_lib_ui(); //!< Ctor
30  };
31
32  //! Static object of type init_lib_ui that is initialised
33  //! before main is run and thus, ui::init is called
34  static init_lib_ui init_obj_ui;
35
36  //! Manipulator class to manipulate UI functions
37  /*!
38   Objects of this type would be used instead of an enum
39   to avoid conflicts with int
40   Every manipulator object is identified by its index while
41   static index indicates the index to be assigned to the next
42   manipulator
43  */
44  class manipulator
45  {
46      static int index; //!< index of a new manipulator object
47      int own_index;    //!< index of current manipulator
48
49      public:
50          manipulator(); //!< Ctor; assigns index
51          int operator==(manipulator); //!< Returns 1 if indexes are same
52  };
53
54  //! Class containing basic UI functions and attributes
55  class ui
56  {
57      ui();   //!< Private ctor; object of this class shouldn't be created
58      public:
59
60          //! Specifies the directions for modifying frame, etc.
61          enum dir
62          {
63              left = 1,
64              top = 2,
65              right = 4,
66              bottom = 8,
67              all = 16 //!< When all sides need to be modified
68          };
69          static int scr_height; //!< Height of screen
70          static int scr_width; //!< Width of screen
71          static void init(); //!< Sets all static variables
72          static void clrscr(); //!< Clears the contents off the screen
73          static int tcolor; //!< text color
74          static int bcolor; //!< background color
75          static manipulator endl; //!< End line and move cursor to next line
76          static manipulator centeralign; //!< Center align
77          static manipulator rightalign; //!< Right align
78
79          //! This func is called when new is unable to allocate memory
80          static void my_new_handler();
81  };
82
83  //! Represents a coordinate
```

```
84  struct coord
85  {
86      int x;   //!< x coordinate
87      int y;   //!< y coordinate
88
89      coord(int = 1,int = 1); //!< Sets the coordinate
90      coord & operator+=(coord);
91      coord & operator-=(coord);
92      coord operator+(coord);
93      coord operator-(coord);
94  };
95
96  //! Represents the node of a list representing the layout
97  /*!
98   Represents all the information of an element that will be
99   printed on the screen. Also points to the next element of the
100  screen that will be printed next to it
101 */
102 class list_layout_node
103 {
104     list_layout_node *next;      //!< Pointer to next node
105     coord pos;                   //!< Position where to print
106     int tcolor;                  //!< Text colour
107     int bcolor;                  //!< Background colour
108     char str[100];               //!< String to print
109
110     //! How to print the string; mainly for passwords
111     int print_type;
112
113     public:
114         list_layout_node();      //!< Ctor
115         ~list_layout_node();     //!< Dtor
116
117         //!@{ Setter functions
118         void setnext(list_layout_node *);
119         void setpos(coord);
120         void settcolor(int);
121         void setbcolor(int);
122         void setstr(const char *);
123         void setprint_type(int);
124         //!@}
125
126         //!@{ Getter functions
127         list_layout_node * getnext();
128         coord getpos();
129         int gettcolor();
130         int getbcolor();
131         const char * getstr();
132         int getprint_type();
133         //!@}
134
135         //! Used to distinguish will be printed i.e.
136         //! as is or hidden (as passwords)
137         enum print_types
138         {
139             DEFAULT,
140             PASSWORD
141         };
142 };
```

```
143
144    //! A node of the representation of string as a linked list
145    struct string_node
146    {
147        string_node *next;   //!< Pointer to next node
148        string_node *prev;   //!< Pointer to previous node
149        char data;           //!< Character stored in string
150
151        string_node();       //!< Ctor
152    };
153
154    //! Represents all interactive information
155    /*!
156     Basically a parent class of all the classes that
157     represent the elements of the layout the user can
158     interact with.
159     Used so that all those elements can be clubbed together
160     and the input be taken.
161    */
162    class interactive : public list_layout_node
163    {
164        interactive *prev;       //!< ptr to previous node
165        interactive *next;       //!< ptr to next node
166        int offset;              //!< offset to y position when printing
167        public:
168            interactive();       //!< Ctor
169            ~interactive();      //!< Dtor
170
171            //! Empty input function that will be overridden by children
172            /*!
173             \param offset The offset to y position
174             \return Action that was performed by the user
175            */
176            virtual int input(int offset);
177
178            //! Setter function
179            void setoffset(int);
180
181            //! Getter function
182            int getoffset();
183
184            //! Actions that are performed by user; returned from input func.
185            enum actions
186            {
187                GOTONEXT,
188                GOTOPREV,
189                CLICKED,
190                BACK //!< When shift—bckspc is pressed
191            };
192
193            //! Keys that user can press to navigate the form
194            enum keys
195            {
196                TAB,
197                ENTER,
198                BACKSPACE,
199                SHIFT_BACKSPACE,
200                SHIFT_TAB,
201                HOME,
```

```
202              END,
203              DELETE,
204              UP,
205              DOWN,
206              LEFT,
207              RIGHT
208          };
209
210          //! Gets key from user and returns code
211          /*
212           \return Keyname corresponding to enum keys
213          */
214          static int getkey();
215  };
216
217  //! Represents a text box
218  /*!
219   Inherits from interactive as a text box can be interacted
220   with. Gets data from user and stores it as a string that
221   can be further converted to the required data type
222  */
223  class text_box : public interactive
224  {
225      //! Represents if the data entered in the text box
226      //! should be displayed as is or replaced with asterisks
227      int is_password;
228
229      public:
230          text_box(); //!< Ctor
231
232          //! Takes input and returns user action
233          /*!
234           /param offset Offset of y coordinate to print
235           /return Action performed by user
236          */
237          int input(int offset = 0);
238
239          //! Prints string represented by a linked list
240          /*
241           Takes in the head pointer of the linked list
242           string and prints the string by iterating through
243           the list. Has no other side effects.
244           /param head ptr to head of the linked list
245          */
246          void print_str(string_node *head);
247
248          //! Setter function
249          void setis_password(int);
250  };
251
252  //! Represents a button that can be clicked
253  /*!
254   Inherits from interactive as a button can be interacted with.
255   A user can click the button while it's input function is
256   running which will return the user action
257  */
258  class button : public interactive
259  {
260      int tcolor_selected; //!< tcolor when selected
```

```cpp
261         int bcolor_selected; //!< bcolor when seilected
262
263     public:
264         button(); //!< Ctor
265
266         //!@{ Setter functions
267         void settcolor_selected(int);
268         void setbcolor_selected(int);
269         //!@}
270
271         //!@{ Getter functions
272         int gettcolor_selected();
273         int getbcolor_selected();
274         //!@}
275
276         //! Input function
277         /*!
278          Effectively allows the button to be clicked
279          /param offset Offset of y coordinate to print
280          /return Action performed by the user
281         */
282         int input(int offset = 0);
283
284         //! Prints the button
285         /*!
286          /param isselected Indicates if button is selected or not
287         */
288         void print(int isselected = 0);
289 };
290
291 //! Represents the layout of the page
292 /*!
293  Incorporates elements like simple nodes as well as other
294  interactive elements. This layout can be contained within
295  a specific height and the overflowing content can reached
296  by scrolling which is also implemented here.
297 */
298 class list_layout
299 {
300     //!@{ Pointers to implement a linked list to elements
301     list_layout_node *head; //!< ptr to head node
302     list_layout_node *current; //!< ptr to current node
303     //!@}
304
305     coord corner_top_left; //!< top left corner of container
306
307     /*!
308      Following are used as temporary placeholders till data
309      is written to the nodes
310     */
311     ///!@{
312     coord pos;
313     int tcolor;
314     int bcolor;
315     int tcolor_selected;
316     int bcolor_selected;
317     int tcolor_input;
318     int bcolor_input;
319     ///!@}
```

24

```cpp
320
321        //!@{ For scrolling implementation
322        int height; //!< Height of the layout
323        int width; //!< Width of the layout
324        int lines_scrolled; //!< Lines currently scrolled
325        //!@}
326
327        //! For better verbosity at internal level
328        enum print_modes
329        {
330            DISPLAY,
331            HIDE
332        };
333
334        //! Prints the layout
335        /*!
336         Prints the layout by iterating through the internal
337         linked list maintained. Has no other side effects
338         /param print_mode How to print the data
339        */
340        void print(int print_mode = DISPLAY);
341    public:
342        list_layout(); //!< Ctor
343
344        //!@{ Set an element (node)
345        list_layout& operator<<(coord); //!< Set coord of node
346
347        //! Set data held by the node
348        list_layout& operator<<(const char *);
349        //!@}
350
351        //! Set a text box
352        /*!
353         Sets a text box at the position indicated by pos and
354         returns a pointer to it
355         /param pos Position at which to set text box
356         /param is_pass If the text box has a password, set to 1
357         /return pointer to the text box set (casted to interactive *)
358        */
359        interactive * settext_box(coord pos, int is_pass = 0);
360
361        //! Set a button
362        /*!
363         Sets a button at the position indicated by pos and
364         returns a pointer to it
365         /param pos Position at which to set the button
366         /param txt The text the button displays
367        */
368        interactive * setbutton(coord pos, const char *txt);
369
370        //!@{ Setter functions
371        void settcolor(int);
372        void setbcolor(int);
373        void settcolor_selected(int);
374        void setbcolor_selected(int);
375        void settcolor_input(int);
376        void setbcolor_input(int);
377        void setcorner_top_left(coord);
378        void setheight(int);
```

25

```
379        void setwidth(int);
380        void setlines_scrolled(int);
381        void setpos(coord);
382        //!@}
383
384        //!@{ Getter functions
385        int getheight();
386        int getwidth();
387        int getlines_scrolled();
388        coord getpos();
389        coord getcorner_top_left();
390        //!@}
391
392        void display(); //!< Display the layout
393        void hide(); //!< Hide the layout
394        void clear(); //!< Deletes contents of the layout
395 };
396
397 //! Represents a border
398 /*!
399  Basically represents a border with characters that can be
400  customised to suit the requirements.
401 */
402 class frame
403 {
404     char border_chars[8];   //!< chars used to draw border
405     int tcolor;             //!< text color
406     int bcolor;             //!< background color
407
408     //! Represents what part of frame is visible.
409     int sides_visibility[8];
410     int frame_visibility;   //!< Frame visible or not
411     coord corner_top_left;  //!< coord of top left corner
412
413     //!@{These include the border characters too
414     int height;             //!< height
415     int width;              //!< width
416     //!@}
417
418     //! Internal pmt used by operator<<
419     int state;
420
421     //! Sets the visibility of the side
422     /*!
423      /param side Specifies the side using ui::dir
424      /param visib Set the visibility of the side
425     */
426     void setside_visibility(int side, int visib);
427
428     //! Converts the ui::dir code into internally usable code
429     int convert(int);
430
431     //! Prints the frame
432     /*!
433      /param f_visib If 1, frame is printed; hidden if it's 0
434     */
435     void print(int f_visib = 1);
436
437     public:
```

26

```cpp
438
439            //! Used to set the visibility mode of the frame
440            /*
441             all: ───────
442                   |     |
443                  ───────
444             nosides: ───────
445
446                      ───────
447            */
448            enum visibility_modes
449            {
450                all = 1,
451                nosides = 2
452            };
453
454            //! Ctor
455            /*!
456             /param corner_top_left Top left corner of frame
457             /param width Width of the frame
458             /param height Height of the frame
459            */
460            frame(coord corner_top_left = coord(1,1), int width =
461            ui::scr_width, int height = ui::scr_height − 1);
462
463            void display(); //!< Display the frame
464            void hide();    //!< Hides the frame
465
466            //! Sets the visibility mode of the frame
467            void setvisibility_mode(int);
468
469            //!@{ operator<<
470            frame & operator<<(int); //!<Sets state
471
472            //! Sets border_char according to state
473            frame & operator<<(char);
474            //!@}
475
476            //!@{ Getter functions
477            int getheight();
478            int getwidth();
479            coord getcorner_top_left();
480
481            //! Returns 1 if visible; 0 = not visible
482            int getframe_visibility();
483            int gettcolor();
484            int getbcolor();
485            char getborder_char(int);
486            int getside_visibility(int);
487            //!@}
488
489            //!@{ Setter functions
490            void setheight(int);
491            void setwidth(int);
492            void settcolor(int);
493            void setbcolor(int);
494            void setcorner_top_left(coord);
495            //!@}
496    };
```

```
497
498   //! Info related to a text box
499   /*!
500    Stores information related to a text box
501    Such as what type to convert it's data to
502    and where to store it
503   */
504   struct info_tbox
505   {
506       text_box * tbox;    //!< ptr to text_box whose info is stored
507
508       //! Data type to convert the string stored in text box to
509       int type;
510       void * data_store;  //!< Where to store converted data
511
512       /*!
513        A validation function that's used to validate the
514        string stored in the text box to see if it is of
515        the required type before converting it.
516        /param str The string to validate
517        /param return 1, if string is validate; 0, otherwise
518       */
519       int (*validator)(const char *str);
520
521       //! The data types the string stored in text box represents
522       /*!
523        Whenever a text box is set, the pointer to the place where
524        final data has to be stored is converted to a void* and
525        the data type is stored.
526        So, void* in different cases is:
527
528        data type       | What void* was
529        ─────────────── | ───────────────────
530        INT             | int *
531        LONG            | long *
532        UNSIGNED_LONG   | unsigned long *
533        STRING          | char *
534        CHAR            | char *
535        DOUBLE          | double *
536        FLOAT           | float *
537        PASSWORD        | char *
538       */
539       enum data_types
540       {
541           INT,
542           LONG,
543           UNSIGNED_LONG,
544           STRING,
545           CHAR,
546           DOUBLE,
547           FLOAT,
548           PASSWORD,
549           OTHER //!< Not supported at the moment
550       };
551
552       info_tbox();    //!< Ctor
553
554       //! Sets data to the data_store
555       /*!
```

```
556        Gets the string stored in the text box, validates
557        it using the validation function and then converts
558        the string to the required data type and stores it in
559        the required space
560        /return 1 on success, 0 on invalid data
561      */
562      int setdata();
563 };
564
565 /*!
566  Contains default validation functions of type
567  int f(char *)
568  that take in a string and return 1 if the string
569  is valid and 0, otherwise
570 */
571 class validation
572 {
573     validation(); //!< Object of this class is not allowed
574     public:
575
576         //!@{ Default validation functions
577         static int vint(const char *);
578         static int vlong(const char *);
579         static int vunsigned_long(const char *);
580         static int vstring(const char *);
581         static int vchar(const char *);
582         static int vdouble(const char *);
583         static int vfloat(const char *);
584         //!@}
585
586         /*!
587          Get the default validator function for the type
588          specified. If func is not NULL, returns default
589          function, else returns v
590         */
591         static validator_f getvalidator(int type,
592                                 validator_f func);
593 };
594
595 /*!
596  Represents a line with the three strings depiciting
597  left, middle and right aligned stuff respectively
598 */
599 struct line
600 {
601     //!@{ Parts of the line
602     char left[100];   //!< left aligned
603     char middle[100]; //!< centre aligned
604     char right[100];  //!< right aligned
605     //!@}
606
607     int width;  //!< width of line
608     int tcolor; //!< text color
609     int bcolor; //!< background color
610     coord corner_top_left; //!< coord of top left corner
611
612     line(); //!< Ctor
613     void display(); //!< Display the line
614     void hide();    //!< Hide the line
```

```cpp
615        void clear();    //!< Delete the data stored
616
617      private:
618          void print(int); //!< Print the line according to arg
619  };
620
621  /*!
622   Default Back function for use in the class box.
623   Can't declare it as member function as member functions
624   are not inherently addresses and setting it as a member function
625   was causing unsolvable problems
626  */
627  int default_back_func();
628
629  //! A box that has a border and a layout
630  /*!
631   Basically incorporates all the elements into a single
632   entity that the user will interact with.
633   Basically looks like
634   ─────────────────  <── Frame
635   | ───────────── |
636   | |             <────────Layout (No border)
637   | |             | |
638   | ───────────── <─────Padding (between layout and frame)
639   ─────────────────
640  */
641  class box
642  {
643      int height;      //!< Height of the box
644      int width;       //!< Width of the box
645      int padding;     //!< Padding between frame and layout
646
647      /*!
648       Wraps a string with specified number of characters
649       in each line
650       /param str String to wrap. Will be modified
651       /param length Number of chars in a line
652       /param return_one_line Sets string to have only one line
653       /return Number of lines after wrapping
654      */
655      int wrap(char str[], int length, int return_one_line = 0);
656
657      //! Sets the tbox
658      /*!
659       Sets the textbox in the layout and also stores the
660       correpsonding data in a tbox that is stored in the array
661       /param data_type Type of data in text box
662       /param ptr Pointer to the data store to set in tbox
663      */
664      void set_tbox(int data_type, void *ptr);
665
666      //!@{ Lists of interactives and text boxes
667      interactive * list_interactive[30];
668      info_tbox list_tbox[30];
669      int index_interactive; //!< Index of element to set next
670      int index_tbox; //!< Index of element to set next
671      //!@}
672
673      //! Clicking this button exits the loop
```

```
674         button * exit_btn;
675
676         //!@{ Toggles that help setting required info in layout
677         int center_toggle;
678         int default_toggle;
679         int right_toggle;
680         int header_toggle;
681         int footer_toggle;
682         int password_toggle;
683         //!@}
684
685         char default_text[100]; //!< Default text to set in textbox
686
687         /*!
688          A temporary variable that stores validator func till it
689          is stored in the required place.
690         */
691         int (*temp_validator)(const char *);
692
693         //!@{ Header and footer
694         line header;
695         line footer;
696         //!@}
697
698         /*!
699          The function is called when the user performs a back func
700          while interacting with any interactive
701          /return 1, if loop exits on back; 0, if it does nothing
702         */
703         int (*back_func)();
704
705     protected:
706         coord pos_pointer;  //!< Pos of pointer in box
707         list_layout layout; //!< Layout in which data is stored
708         coord corner_top_left; //!< Coord of top left corner
709
710     public:
711
712         //!@{ Manipulators can be used to alter function of <<
713         static manipulator setheader;
714         static manipulator setfooter;
715         static manipulator setpassword;
716         //!@}
717
718         frame f;     //!< Border of the box
719
720         //! Ctor
721         /*!
722          Initialises all the variables of the class
723          /param corner_top_left The top left corner
724          /param width Width of box (includes border)
725          /param height Height of box (includes border)
726         */
727         box(coord corner_top_left = coord(1,1),
728             int width = ui::scr_width,
729             int height= ui::scr_height − 1);
730
731         //!@{ Getter functions
732         coord getcorner_top_left();
```

```cpp
733         int getheight();
734         int getwidth();
735         int getpadding();
736         //!@}
737
738         //!@{ Setter functions
739         void setcorner_top_left(coord);
740         void setheight(int);
741         void setpadding(int);
742         void settcolor(int);
743         void setbcolor(int);
744         void settcolor_selected(int);
745         void setbcolor_selected(int);
746         void settcolor_input(int);
747         void setbcolor_input(int);
748         void setback_func( int(*f)(void) );
749         //!@}
750
751         //!@{ operator<< is used for adding data to the box's
752         //!   layout that will be printed
753         box & operator<<(char *);
754         box & operator<<(char);
755         box & operator<<(int);
756         box & operator<<(long);
757         box & operator<<(unsigned long);
758         box & operator<<(double);
759         box & operator<<(float);
760         box & operator<<(manipulator);
761         //!@}
762
763         //!@{ operator>> is used for basically setting a text
764         //!   box at the place where pos_pointer is currently
765         //!   at
766         box & operator>>(char *&);
767         box & operator>>(char &);
768         box & operator>>(int &);
769         box & operator>>(long &);
770         box & operator>>(unsigned long &);
771         box & operator>>(double &);
772         box & operator>>(float &);
773         box & operator>>(manipulator);
774
775         //! Using this before another >> will set this func
776         //! as the validator of that text box
777         box & operator>>(int (*)(const char *));
778         //!@}
779
780         void setexit_button(char *);
781
782         //!@{ Sets default for the next text box and
783         //!   clears it after the next text box has been
784         //!   set
785         void setdefault(char *);
786         void setdefault(char);
787         void setdefault(int);
788         void setdefault(long);
789         void setdefault(unsigned long);
790         void setdefault(double);
791         void setdefault(float);
```

```
792            //!@}
793
794            /*!
795             Sets the box to loop, effectively enabling
796             all the text boxes and buttons. Also enables
797             scrolling
798            */
799            void loop();
800
801            void display(); //!< Display the box
802            void hide();    //!< Hide the box
803            void clear();   //!< Delete the contents of the box
804
805            //!@{ Functions to set header and footer
806            void setheader_tcolor(int); //!< set header color
807            void setfooter_tcolor(int); //!< set footer color
808            void clear_header(); //!< Delete contents of header
809            void clear_footer(); //!< Delete contents of footer
810            //!@}
811    };
812
813    #endif /* UI_HPP */
```

1. code/iface3.cpp

```cpp
1   #include <fstream.h>
2   #include "base.hpp"
3   #include "iface.hpp"
4   #include "hosp.hpp"
5   #include "emp.hpp"
6
7   void interface::employee_management()
8   {
9       const int menu_corner_top_left_y = 5;
10      coord c(ui::scr_width * 0.2, menu_corner_top_left_y);
11      int ch;
12      while(1)
13      {
14          interface::clear_error();
15          box menu(c, ui::scr_width * 0.6, ui::scr_height - 6 );
16          menu.settcolor(GREEN);
17          menu << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
18          menu.settcolor(ui::tcolor);
19          menu << "1. View employee data" << ui::endl
20              << "2. Add new employee" << ui::endl
21              << "3. Remove existing employee" << ui::endl
22              << "4. Edit employee data" << ui::endl
23              << "5. Pay salary to individual employee" << ui::endl
24              << "6. Pay salary to all employees" << ui::endl
25              << "7. Back" << ui::endl
26              << ui::endl << "Enter your choice: ";
27          menu.settcolor_input(YELLOW);
28          validate_menu::set_menu_limits(1, 7);
29          menu >> validate_menu::input >> ch;
30          menu << ui::endl;
31          menu.setexit_button("Submit");
32          menu.loop();
33          menu.hide();
34          switch (ch)
35          {
36              case 1:
37              {
38                  emp_mgmt::view_emp();
39                  break;
40              }
41              case 2:
42              {
43                  emp_mgmt::add_emp();
44                  break;
45              }
46              case 3:
47              {
48                  emp_mgmt::remove_emp();
49                  break;
50              }
51              case 4:
52              {
53                  emp_mgmt::edit_emp();
54                  break;
55              }
```

```cpp
56              case 5:
57              {
58                  emp_mgmt::pay_emp();
59                  break;
60              }
61              case 6:
62              {
63                  emp_mgmt::pay_all();
64                  break;
65              }
66              case 7:
67              {
68                  return;
69              }
70          }
71      }
72  }
73
74  void interface::employee_screen(unsigned long id)
75  {
76      void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
               among employee, doctor, nurse and receptionist classes
77      if(temp == NULL)
78      {
79          interface::log_this("interface::employee_screen() : Not enough memory to
                   allocate buffer void * temp = malloc( sizeof(doctor) )");
80          interface::error("Out of memory!! Check log");
81          getch();
82          return;
83      }
84      if(!hospital::get_employee_by_id(id, temp))
85      {
86          interface::error("ID not found or error while reading from file!");
87          getch();
88          free(temp);
89          return;
90      }
91      employee *e = (employee *) temp;
92      const int menu_corner_top_left_y = 5;
93      coord c(ui::scr_width * 0.2, menu_corner_top_left_y);
94      int ch;
95      str heading = "Welcome, ";
96      strcat( heading, e->get_name() );
97      strcat(heading, "!");
98      while(1)
99      {
100         interface::clear_error();
101         box menu(c, ui::scr_width * 0.6, ui::scr_height - 6 );
102         menu.settcolor(GREEN);
103         menu << ui::centeralign << heading << ui::endl << ui::endl;
104         menu.settcolor(ui::tcolor);
105         menu << "1. View profile" << ui::endl
106             << "2. Change login details" << ui::endl
107             << "3. View last 5 transactions" << ui::endl;
108         emp_type type_of_emp = id_to_emp::convert(id);
109         if(type_of_emp == RECEPTIONIST)
110         {
111             menu << "4. Manage patients" << ui::endl
112                 << "5. Exit" << ui::endl;
```

```cpp
113                }
114            else
115            {
116                menu << "4. Exit" << ui::endl;
117            }
118            menu << ui::endl << "Enter your choice: ";
119            menu.settcolor_input(YELLOW);
120            if(type_of_emp == RECEPTIONIST)
121            {
122                validate_menu::set_menu_limits(1, 5);
123            }
124            else
125            {
126                validate_menu::set_menu_limits(1, 4);
127            }
128            menu >> validate_menu::input >> ch;
129            menu << ui::endl;
130            menu.setexit_button("Submit");
131            menu.loop();
132            menu.hide();
133            switch(ch)
134            {
135                case 1:
136                {
137                    if( !emp_mgmt::view_emp(id) )
138                    {
139                        interface::error("Failed to display profile!");
140                        getch();
141                    }
142                    break;
143                }
144                case 2:
145                {
146                    int ch;
147                    while(1)
148                    {
149                        box menu3(c, ui::scr_width * 0.6, ui::scr_height - 6);
150                        menu3.settcolor(GREEN);
151                        menu3 << ui::centeralign << "Change login details" << ui::
                                endl << ui::endl;
152                        menu3.settcolor(WHITE);
153                        menu3 << "1. Change User ID" << ui::endl
154                                << "2. Change Password" << ui::endl
155                                << "3. Back" << ui::endl
156                                << "Enter your choice: ";
157                        menu3.settcolor(ui::tcolor);
158                        menu3.settcolor_input(YELLOW);
159                        validate_menu::set_menu_limits(1, 3);
160                        menu3 >> validate_menu::input >> ch;
161                        menu3 << ui::endl;
162                        menu3.setexit_button("Submit");
163                        menu3.loop();
164                        menu3.hide();
165                        switch(ch)
166                        {
167                            case 1:
168                            {
169                                str new_username;
170                                box menu4( menu3.getcorner_top_left(), menu3.getwidth
```

36

```
                                        (), menu3.getheight() );
171                         menu4.settcolor(GREEN);
172                         menu4 << ui::centeralign << "Change login details" <<
                                 ui::endl << ui::endl;
173                         menu4.settcolor(WHITE);
174                         menu4 << "Change User ID" << ui::endl;
175                         menu4.settcolor(ui::tcolor);
176                         menu4 << "User ID: ";
177                         menu4.setdefault( e->account.get_username() );
178                         menu4.settcolor_input(YELLOW);
179                         menu4 >> new_username;
180                         menu4.setexit_button("Submit");
181                         menu4.setback_func(back_func::set_backbit);
182                         menu4.loop();
183                         menu4.hide();
184                         if(back_func::backbit)
185                         {
186                             back_func::backbit = 0;
187                             break;
188                         }
189                         e->account.set_username(new_username);
190                         const int notice_height = 10;
191                         box notice( menu4.getcorner_top_left(), menu4.
                                 getwidth(), notice_height );
192                         notice.settcolor(GREEN);
193                         notice << ui::centeralign << "Change login details"
                                 << ui::endl << ui::endl;
194                         if( !hospital::write_employee(temp) )
195                         {
196                             notice.settcolor(RED);
197                             notice << "Failed to write new user ID to file!
                                     Check log" << ui::endl;
198                         }
199                         else
200                         {
201                             notice.settcolor(GREEN);
202                             notice << "User ID changed successfully!" << ui::
                                     endl;
203                         }
204                         notice.setexit_button("Back");
205                         notice.loop();
206                         notice.hide();
207                         goto loop_exit;
208                     }
209                     case 2:
210                     {
211                         str curr_pwd, new_pwd;
212                         for(int i = 0; i < 3; ++i)
213                         {
214                             box menu4( menu3.getcorner_top_left(), menu3.
                                     getwidth(), menu3.getheight() );
215                             menu4.settcolor(GREEN);
216                             menu4 << ui::centeralign << "Change login details
                                     " << ui::endl << ui::endl;
217                             menu4.settcolor(WHITE);
218                             menu4 << "Change Password" << ui::endl;
219                             menu4.settcolor(ui::tcolor);
220                             menu4 << "Enter current password: ";
221                             menu4.settcolor_input(YELLOW);
```

37

```
222                          menu4 >> box::setpassword >> curr_pwd;
223                          menu4.setexit_button("Submit");
224                          menu4.setback_func(back_func::set_backbit);
225                          menu4.loop();
226                          menu4.hide();
227                          if(back_func::backbit)
228                          {
229                              break;
230                          }
231                          if( e->account.login(curr_pwd) )
232                          {
233                              interface::clear_error();
234                              break;
235                          }
236                          interface::error("Invalid password!! Try again...
                             ");
237                      }
238                      if(back_func::backbit)
239                      {
240                          back_func::backbit = 0;
241                          break;
242                      }
243                      if(i == 3)
244                      {
245                          const int notice_height = 10;
246                          box notice( menu3.getcorner_top_left(), menu3.
                             getwidth(), notice_height);
247                          notice.settcolor(GREEN);
248                          notice << ui::centeralign << "Change login
                             details" << ui::endl << ui::endl;
249                          notice.settcolor(RED);
250                          notice << "Since you entered the wrong password
                             too many times, you have been logged out. "
251                                  << "Hit the button below to exit the
                                     program." << ui::endl << ui::endl;
252                          notice.setexit_button("Exit");
253                          notice.loop();
254                          notice.hide();
255                          free(temp);
256                          return;
257                      }
258                      box menu5( menu3.getcorner_top_left(), menu3.getwidth
                         (), menu3.getheight() );
259                      menu5.settcolor(GREEN);
260                      menu5 << ui::centeralign << "Change login details" <<
                          ui::endl << ui::endl;
261                      menu5.settcolor(WHITE);
262                      menu5 << "Change Password" << ui::endl;
263                      menu5.settcolor(ui::tcolor);
264                      menu5 << "Enter new password: ";
265                      menu5.settcolor_input(YELLOW);
266                      menu5 >> box::setpassword >> new_pwd;
267                      menu5.setexit_button("Submit");
268                      menu5.setback_func(back_func::set_backbit);
269                      menu5.loop();
270                      menu5.hide();
271                      if(back_func::backbit)
272                      {
273                          back_func::backbit = 0;
```

```
274                                        break;        //At the "Enter new password" page,
                                                          when shift+bkspc is pressed, control will go
                                                          back to "Change login details" menu.
275                                    }
276                                    e->account = userid( e->account.get_username(),
                                           new_pwd );
277                                    const int notice2_height = 13;
278                                    box notice2( menu3.getcorner_top_left(), menu3.
                                           getwidth(), notice2_height );
279                                    notice2.settcolor(GREEN);
280                                    notice2 << ui::centeralign << "Change login details"
                                           << ui::endl << ui::endl;
281                                    if( !hospital::write_employee(temp) )
282                                    {
283                                        notice2.settcolor(RED);
284                                        notice2 << "Failed to write new password to file!
                                               Check log" << ui::endl;
285                                    }
286                                    else
287                                    {
288                                        notice2.settcolor(GREEN);
289                                        notice2 << "Password changed successfully!" << ui
                                               ::endl;
290                                    }
291                                    notice2.settcolor(ui::tcolor);
292                                    notice2 << "Please logout and login again by exiting
                                           the program and restarting it." << ui::endl
293                                            << "Press the button below to exit the
                                                   program." << ui::endl;
294                                    notice2.setexit_button("Exit");
295                                    notice2.loop();
296                                    notice2.hide();
297                                    free(temp);
298                                    return;
299                                }
300                            case 3:
301                            {
302                                goto loop_exit;
303                            }
304                        }
305                    }
306                loop_exit:
307                break;
308            }
309        case 3:
310        {
311            transaction * t = e->get_last_5_transactions();
312            if( t == NULL )
313            {
314                interface::error("Error while reading or writing to file!
                       Check log");
315                getch();
316                break;
317            }
318            coord c2(1, 4);
319            box menu2(c2, (ui::scr_width / 2), ui::scr_height - 5);
320            box sidemenu(( c2 + coord((ui::scr_width / 2) - 1, 0)), (ui::
                   scr_width / 2) + 1, ui::scr_height - 5);
321            menu2.f << ( ui::top | ui::left ) << (char)204
```

```cpp
322                      << ( ui::bottom | ui::left ) << (char)204
323                      << ( ui::top | ui::right ) << (char)203
324                      << ( ui::bottom | ui::right ) << (char)202;
325                  menu2.f.display();
326                  sidemenu.f << ( ui::top | ui::left ) << (char)203
327                          << ( ui::bottom | ui::left ) << (char)202
328                          << ( ui::top | ui::right ) << (char)185
329                          << ( ui::bottom | ui::right ) << (char)185;
330                  sidemenu.f.display();
331                  menu2.settcolor(GREEN);
332                  menu2 << ui::centeralign << "View last 5 transactions" << ui::
                        endl << ui::endl;
333                  menu2.settcolor(ui::tcolor);
334                  for(int i = 0; i < 5; ++i)
335                  {
336                      if( t[i].amount == 0 && !strcmp(t[i].reason, "NA") &&
337                          t[i]._date.day == 0 && t[i]._date.month == 0 && t[i].
                            _date.year == 0
338                          && t[i]._time.hour == 25 && t[i]._time.minute == 0 && t[i
                            ]._time.second == 0 )
339                      {
340                          break;
341                      }
342                      if(i < 3)
343                      {
344                          menu2 << i + 1 << ". " << t[i]._date << ", " << t[i].
                                _time << ui::endl
345                              << "Amount: " << t[i].amount << ui::endl
346                              << "Reason: " << t[i].reason << ui::endl;
347                      }
348                      else
349                      {
350                          sidemenu << i + 1 << ". " << t[i]._date << ", " << t[i].
                                _time << ui::endl
351                                  << "Amount: " << t[i].amount << ui::endl
352                                  << "Reason: " << t[i].reason << ui::endl;
353                      }
354                  }
355                  free(t);
356                  if(i <= 3)
357                  {
358                      menu2.setexit_button("Back");
359                      menu2.loop();
360                  }
361                  else
362                  {
363                      sidemenu.setexit_button("Back");
364                      sidemenu.loop();
365                  }
366                  menu2.hide();
367                  sidemenu.hide();
368                  window.f.display();
369                  break;
370              }
371          case 4:
372              {
373                  if(type_of_emp == RECEPTIONIST)
374                  {
375                      interface::patient_management();
```

```
376                        break;
377                    }
378                    else
379                    {
380                        free(temp);
381                        return;
382                    }
383                }
384                case 5:
385                {
386                    free(temp);
387                    return;
388                }
389            }
390        }
391    }
392
393    emp_mgmt::emp_mgmt()
394    {}
395
396    void emp_mgmt::view_emp()
397    {
398        const int menu2_height = 10;
399        box menu2( coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, menu2_height);
400        menu2.settcolor(GREEN);
401        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
402        menu2.settcolor(WHITE);
403        menu2 << "View employee data" << ui::endl;
404        menu2.settcolor(ui::tcolor);
405        menu2 << "Enter employee's id: ";
406        unsigned long id;
407        menu2.settcolor_input(YELLOW);
408        menu2 >> id;
409        menu2 << ui::endl;
410        menu2.setexit_button("Submit");
411        menu2.setback_func(back_func::set_backbit);
412        menu2.loop();
413        menu2.hide();
414        if(back_func::backbit)
415        {
416            back_func::backbit = 0;
417            return;
418        }
419        view_emp(id);
420    }
421
422    int emp_mgmt::view_emp(unsigned long id)
423    {
424        void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
                 among employee, doctor, nurse and receptionist classes
425        if(temp == NULL)
426        {
427            interface::log_this("emp_mgmt::view_emp(int) : Not enough memory to
                     allocate buffer void * temp = malloc( sizeof(doctor) )");
428            interface::error("Out of memory!! Check log");
429            getch();
430            return 0;
431        }
432        if(!hospital::get_employee_by_id(id, temp))
```

```
433         {
434             interface::error("ID not found or error while reading from file!");
435             getch();
436             free(temp);
437             return 0;
438         }
439         employee *e = (employee *) temp;
440         box menu3( coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, ui::scr_height
                  − 6 );
441         menu3.settcolor(GREEN);
442         menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
443         menu3.settcolor(WHITE);
444         menu3 << "Employee Details: " << ui::endl;
445         menu3.settcolor(ui::tcolor);
446         menu3 << "ID: " << e->get_id() << ui::endl;
447         menu3 << "Name: " << e->get_name() << ui::endl;
448         menu3 << "Age: " << e->get_age() << ui::endl;
449         menu3 << "Sex: " << (sex)e->get_sex() << ui::endl;
450         menu3 << "Date of Birth: " << e->get_dob() << ui::endl;
451         menu3 << "Address: " << e->get_address() << ui::endl;
452         menu3 << "Phone no.: " << e->get_phone() << ui::endl;
453         menu3 << "Salary: " << e->get_salary() << ui::endl;
454         menu3 << "Shift timings: Starts − " << e->get_shift(START) << ui::endl;
455         menu3 << "────────────: Ends − " << e->get_shift(END) <<ui::endl;
456         switch( id_to_emp::convert( e->get_id() ) )
457         {
458             case INVALID:   //Test this case, menu3.hide() not working properly
459             {
460                 menu3.clear();
461                 int menu3_height = 9;
462                 menu3.setheight(menu3_height);
463                 menu3.settcolor(GREEN);
464                 menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::
                        endl;
465                 menu3.settcolor(WHITE);
466                 menu3 << "Employee Details: " << ui::endl;
467                 menu3.settcolor(RED);
468                 menu3 << "Invalid ID!!" << id_to_emp::convert( e->get_id() );
469                 menu3.settcolor(ui::tcolor);
470                 menu3.setexit_button("Back");
471                 menu3.loop();
472                 menu3.hide();
473                 break;
474             }
475             case OTHERS:
476             case RECEPTIONIST:  //there are no extra data members in class
                    receptionist
477             {
478                 menu3.setexit_button("Back");
479                 menu3.loop();  //   menu3.clear();  int w = window.getwidth(), m =
                        menu3.getwidth(); menu3<<w<<' '<<m;  getch();
480                 menu3.hide();
481                 break;
482             }
483             case DOCTOR:
484             {
485                 doctor *d = (doctor *)temp;
486                 menu3.hide();
487                 menu3.setcorner_top_left( coord( 1, menu3.getcorner_top_left().y ) );
```

```cpp
488                menu3.display();
489                menu3.f << ( ui::top | ui::left ) << (char)204
490                        << ( ui::bottom | ui::left ) << (char)204;
491                menu3.f.display();
492                box sidemenu( menu3.getcorner_top_left() + coord( menu3.getwidth() -
                       1, 0 ), ( ui::scr_width - menu3.getwidth() + 1 ),  menu3.getheight
                       () );
493                sidemenu.f << ( ui::top | ui::left ) << (char)203
494                        << ( ui::bottom | ui::left ) << (char)202
495                        << ( ui::top | ui::right ) << (char)185
496                        << ( ui::bottom | ui::right ) << (char)185;
497                sidemenu.f.display();
498                sidemenu << "Speciality(s)" << ui::endl;
499                for(int i = 0; i < 2 && d->get_speciality()[i] <= GEN; ++i)
500                {
501                    sidemenu << i + 1 << ". " << (body_parts)d->get_speciality()[i]
                           << ui::endl;
502                }
503                if(!i)
504                {
505                    sidemenu << "None" << ui::endl;
506                }
507                sidemenu << "Patients currently under care:" << ui::endl;
508                for(i = 0; d->get_patients()[i] && i < 10; ++i)
509                {
510                    sidemenu << i + 1 << ". " << hospital::get_patient_by_id( d->
                           get_patients()[i] ).get_name() << ui::endl;
511                }
512                if(!i)
513                {
514                    sidemenu << "None" << ui::endl;
515                }
516                sidemenu.setexit_button("Back");
517                sidemenu.loop();
518                menu3.hide();
519                sidemenu.hide();
520                window.f.display();
521                break;
522            }
523            case NURSE:
524            {
525                nurse *n = (nurse *)temp;
526                menu3.hide();
527                menu3.setcorner_top_left( coord( 1, menu3.getcorner_top_left().y ) );
528                menu3.display();
529                menu3.f << ( ui::top | ui::left ) << (char)204
530                        << ( ui::bottom | ui::left ) << (char)204;
531                menu3.f.display();
532                box sidemenu( menu3.getcorner_top_left() + coord( menu3.getwidth() -
                       1, 0 ), ( ui::scr_width - menu3.getwidth() + 1 ),  menu3.getheight
                       () );
533                sidemenu.f << ( ui::top | ui::left ) << (char)203
534                        << ( ui::bottom | ui::left ) << (char)202
535                        << ( ui::top | ui::right ) << (char)185
536                        << ( ui::bottom | ui::right ) << (char)185;
537                sidemenu.f.display();
538                sidemenu << "Patients currently under care:" << ui::endl;
539                for(int i = 0; n->get_patients()[i] && i < 5; ++i)
540                {
```

43

```
541                    sidemenu << i + 1 << ". " << hospital::get_patient_by_id( n->
                            get_patients()[i] ).get_name() << ui::endl;
542                }
543                if(!i)
544                {
545                    sidemenu << "None" << ui::endl;
546                }
547                sidemenu.setexit_button("Back");
548                sidemenu.loop();
549                menu3.hide();
550                sidemenu.hide();
551                window.f.display();
552                break;
553            }
554        }
555        free(temp);
556        return 1;
557 }
558
559 void emp_mgmt::add_emp()
560 {
561        int ch;
562        str name, dob_str, adr_hno, adr_street, adr_city, adr_dist, adr_state,
                shift_start_str, shift_end_str, uid, pwd;
563        unsigned sex_choice;
564        Date dob;
565        address adr;
566        phone phn_no;
567        unsigned long salary;
568        Time shift_start, shift_end;
569        int speciality[2];
570        const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
571        const int menu2_width = ui::scr_width * 0.6;
572        menu2:
573        {
574            const int menu2_height = 17;
575            box menu2(menu2_corner_top_left, menu2_width, menu2_height);
576            menu2.settcolor(GREEN);
577            menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                    ;
578            menu2.settcolor(WHITE);
579            menu2 << "Add new employee" << ui::endl;
580            menu2.settcolor(ui::tcolor);
581            menu2 << "Step 1: Select employee type" << ui::endl << ui::endl
582                    << "1. Doctor" << ui::endl
583                    << "2. Nurse" << ui::endl
584                    << "3. Receptionist" << ui::endl
585                    << "4. Others" << ui::endl << ui::endl
586                    << "Enter your choice: ";
587            validate_menu::set_menu_limits(1, 4);
588            menu2.settcolor_input(YELLOW);
589            menu2 >> validate_menu::input >> ch;
590            menu2 << ui::endl;
591            menu2.setexit_button("Submit");
592            menu2.setback_func(back_func::set_backbit);
593            menu2.loop();
594            menu2.hide();
595            if(back_func::backbit)
596            {
```

```
597            back_func::backbit = 0;
598            return;
599        }
600    }
601    menu3:
602    {
603        box menu3( menu2_corner_top_left, menu2_width, ui::scr_height − 6 );
604        menu3.settcolor(GREEN);
605        menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
              ;
606        menu3.settcolor(WHITE);
607        menu3 << "Add new employee" << ui::endl;
608        menu3.settcolor(ui::tcolor);
609        menu3 << "Step 2: Add employee details" << ui::endl << ui::endl;
610        menu3.settcolor_input(YELLOW);
611        menu3 << "Name: ";
612        menu3 >> name;
613        menu3 << "Sex: 1. Male | 2. Female | 3. Transsexual" << ui::endl
614              << "——— Enter your choice: ";
615        validate_menu::set_menu_limits(1, 3);
616        menu3 >> validate_menu::input >> (int)sex_choice;
617        menu3 << "Date of Birth(DD/MM/YYYY): ";
618        menu3 >> hospital::date_validity >> dob_str;
619        menu3 << "Address: " << ui::endl;
620        menu3 << (char)26 << "House no.: ";
621        menu3 >> adr_hno;
622        menu3 << (char)26 << "Street: ";
623        menu3 >> adr_street;
624        menu3 << (char)26 << "City: ";
625        menu3 >> adr_city;
626        menu3 << (char)26 << "District: ";
627        menu3 >> adr_dist;
628        menu3 << (char)26 << "State: ";
629        menu3 >> adr_state;
630        menu3 << "Phone no.: ";
631        menu3 >> phn_no;
632        menu3 << "Salary: ";
633        menu3 >> salary;
634        menu3 << "Shift timings: Starts — (HH:MM:SS)";
635        menu3 >> hospital::time_validity >> shift_start_str;
636        menu3 << "—————————: Ends — (HH:MM:SS)";
637        menu3 >> hospital::time_validity >> shift_end_str;
638        menu3.setexit_button("Submit");
639        menu3.setback_func(back_func::set_backbit);
640        menu3.loop();
641        menu3.hide();
642        if(back_func::backbit)
643        {
644            back_func::backbit = 0;
645            goto menu2;
646        }
647        ——sex_choice;
648        dob = hospital::str_to_date(dob_str);
649        adr = address(adr_hno, adr_street, adr_city, adr_dist, adr_state);
650        shift_start = hospital::str_to_time(shift_start_str);
651        shift_end = hospital::str_to_time(shift_end_str);
652    }
653    menu4:
654
```

```
655     if(ch != 4)
656     {
657         box menu4( menu2_corner_top_left, menu2_width, ui::scr_height − 6 );
658         menu4.settcolor(GREEN);
659         menu4 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                  ;
660         menu4.settcolor(WHITE);
661         menu4 << "Add new employee" << ui::endl;
662         menu4.settcolor(ui::tcolor);
663         menu4.settcolor_input(YELLOW);
664         menu4 << "Step 3: Add login details" << ui::endl << ui::endl;
665         menu4 << "User ID: ";
666         menu4 >> uid;
667         menu4 << "Password: ";
668         menu4 >> box::setpassword >> pwd;
669         menu4 << ui::endl;
670         menu4.setexit_button("Submit");
671         menu4.setback_func(back_func::set_backbit);
672         menu4.loop();
673         menu4.hide();
674     }
675     if(back_func::backbit)
676     {
677         back_func::backbit = 0;
678         goto menu3;
679     }
680     if(ch == 1)
681     {
682         coord c(1, 4);
683         box menu5(c, (ui::scr_width / 2), ui::scr_height − 5);
684         box inp_box(( c + coord((ui::scr_width / 2) − 1, 0)), (ui::scr_width / 2)
                  + 1, ui::scr_height − 5);
685         menu5.f << ( ui::top | ui::left ) << (char)204
686                 << ( ui::bottom | ui::left ) << (char)204
687                 << ( ui::top | ui::right ) << (char)203
688                 << ( ui::bottom | ui::right ) << (char)202;
689         menu5.f.display();
690         inp_box.f << ( ui::top | ui::left ) << (char)203
691               << ( ui::bottom | ui::left ) << (char)202
692               << ( ui::top | ui::right ) << (char)185
693               << ( ui::bottom | ui::right ) << (char)185;
694         inp_box.f.display();
695         menu5 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                  ;
696         menu5.settcolor(WHITE);
697         menu5 << "Add new employee" << ui::endl;
698         menu5.settcolor(ui::tcolor);
699         menu5 << "Step 4: Add doctor details" << ui::endl << ui::endl;
700         menu5 << "Specialization of doctor (max 2)" << ui::endl
701               << "Choose from the following list: " << ui::endl;
702         for(int i = 0; i <= GEN; ++i)
703         {
704             if(i <= 8)
705             {
706                 menu5 << i << ". " << (body_parts)i << ui::endl;
707             }
708             else
709             {
710                 inp_box << i << ". " << (body_parts)i << ui::endl;
```

46

```
711                 }
712             }
713             inp_box.settcolor_input(YELLOW);
714             inp_box << "Enter the number corresponding to the required entry in the 2
                     fields below" << ui::endl;
715             validate_menu::set_menu_limits(BRAIN, GEN);
716             inp_box << (char)26;    inp_box >> validate_menu::input >> speciality[0];
717             inp_box << (char)26;    inp_box >> validate_menu::input >> speciality[1];
718             inp_box << ui::endl;
719             inp_box.setexit_button("Submit");
720             inp_box.setback_func(back_func::set_backbit);
721             inp_box.loop();
722             menu5.hide();
723             inp_box.hide();
724             window.f.display();
725         }
726     if(back_func::backbit)
727     {
728         back_func::backbit = 0;
729         goto menu4;
730     }
731     void * temp = NULL;
732     unsigned long id;
733     switch (ch)
734     {
735         case 1:
736         {
737             doctor x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                     shift_end, speciality[0], speciality[1], uid, pwd);
738             temp = &x;
739             id = x.get_id();
740             break;
741         }
742         case 2:
743         {
744             nurse x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                     shift_end, uid, pwd);
745             temp = &x;
746             id = x.get_id();
747             break;
748         }
749         case 3:
750         {
751             receptionist x(name, sex_choice, dob, adr, phn_no, salary,
                     shift_start, shift_end, uid, pwd);
752             temp = &x;
753             id = x.get_id();
754             break;
755         }
756         case 4:
757         {
758             employee x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                     shift_end);
759             temp = &x;
760             id = x.get_id();
761             break;
762         }
763     }
764     const int notice_height = 12;
```

47

```
765         box notice( menu2_corner_top_left, menu2_width, notice_height );
766         notice.settcolor(GREEN);
767         notice << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
768         if(!hospital::write_employee(temp))
769         {
770             notice.settcolor(RED);
771             notice << "Employee addition unsuccessful!!";
772             notice.setexit_button("Exit");
773             notice.loop();
774             notice.hide();
775             return;
776         }
777         notice << "Employee added successfully!!" << ui::endl;
778         notice.settcolor(WHITE);
779         notice << "Hit the button below to display the details you entered: " << ui::
                endl;
780         notice.settcolor(ui::tcolor);
781         notice << ui::endl;
782         notice.setexit_button("View employee...");
783         notice.loop();
784         notice.hide();
785         view_emp(id);
786     }
787
788     void emp_mgmt::remove_emp()
789     {
790         const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
791         const int menu2_width = ui::scr_width * 0.6;
792         unsigned long id;
793         char ch;
794         menu2:
795         {
796             const int menu2_height = 10;
797             box menu2(menu2_corner_top_left, menu2_width, menu2_height);
798             menu2.settcolor(GREEN);
799             menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                    ;
800             menu2.settcolor(WHITE);
801             menu2 << "Remove existing employee" << ui::endl;
802             menu2.settcolor(ui::tcolor);
803             menu2 << "Enter employee's id: ";
804             menu2.settcolor_input(YELLOW);
805             menu2 >> id;
806             menu2 << ui::endl;
807             menu2.setexit_button("Submit");
808             menu2.setback_func(back_func::set_backbit);
809             menu2.loop();
810             menu2.hide();
811         }
812         if(back_func::backbit)
813         {
814             back_func::backbit = 0;
815             return;
816         }
817         notice:
818         {
819             const int notice_height = 14;
820             box notice(menu2_corner_top_left, menu2_width, notice_height);
821             notice.settcolor(GREEN);
```

48

```
822            notice << ui::centeralign << "Employee Management" << ui::endl << ui::
                  endl;
823            notice.settcolor(WHITE);
824            notice << "Hit the button below to display the details of the employee
                  you want to remove: " << ui::endl;
825            notice.settcolor(ui::tcolor);
826            notice << ui::endl;
827            notice.setexit_button("View employee...");
828            notice.setback_func(back_func::set_backbit);
829            notice.loop();
830            notice.hide();
831        }
832        if(back_func::backbit)
833        {
834            back_func::backbit = 0;
835            goto menu2;
836        }
837        if( !view_emp(id) )
838        {
839            return;
840        }
841        notice2:
842        {
843            const int notice2_height = 14;
844            box notice2( menu2_corner_top_left, menu2_width, notice2_height );
845            notice2.settcolor(GREEN);
846            notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::
                  endl;
847            notice2.settcolor(WHITE);
848            notice2 << "Are you sure you want to remove this employee?(y/n): " << ui
                  ::endl;
849            notice2.settcolor_input(YELLOW);
850            notice2 >> ch;
851            notice2.settcolor(ui::tcolor);
852            notice2 << ui::endl;
853            notice2.setexit_button("Submit");
854            notice2.setback_func(back_func::set_backbit);
855            notice2.loop();
856            notice2.hide();
857        }
858        if(back_func::backbit)
859        {
860            back_func::backbit = 0;
861            goto notice;
862        }
863        if(ch == 'n' || ch == 'N')
864        {
865            return;
866        }
867        const int notice3_height = 14;
868        box notice3( menu2_corner_top_left, menu2_width, notice3_height );
869        notice3.settcolor(GREEN);
870        notice3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
871        notice3.settcolor(RED);
872        str path;
873        switch(id_to_emp::convert(id))
874        {
875            case INVALID:
876                interface::log_this("emp_mgmt::remove_emp() : No file with zero id
```

49

```
                    exists\nFunction aborted");
877                 notice3 << "Invalid ID supplied!! Check log" << ui::endl;
878                 notice3.setexit_button("Back");
879                 notice3.loop();
880                 notice3.hide();
881                 return;
882         case OTHERS:
883             sprintf(path, "employee/%lu", id);
884             break;
885         case DOCTOR:
886             mkdir("employee/doctor");
887             sprintf(path, "employee/doctor/%lu", id);
888             break;
889         case NURSE:
890             mkdir("employee/nurse");
891             sprintf(path, "employee/nurse/%lu", id);
892             break;
893         case RECEPTIONIST:
894             mkdir("employee/receptionist");
895             sprintf(path, "employee/receptionist/%lu", id);
896             break;
897     }
898     int remove_status;
899     str file;
900     strcpy(file, path);
901     strcat(file, "/base.dat");
902     if( remove(file) == -1)
903     {
904         str log_str;
905         sprintf(log_str, "emp_mgmt::remove_emp() : Failed to delete base.dat file
                    of id %lu\nFunction aborted", id);
906         interface::log_this(log_str);
907         notice3 << "Failed to delete file of employee!!" << ui::endl;
908         notice3.setexit_button("Back");
909         notice3.loop();
910         notice3.hide();
911         return;
912     }
913     if( rmdir(path) == -1)
914     {
915         str log_str;
916         sprintf(log_str, "emp_mgmt::remove_emp() : Failed to delete folder of id
                    %lu", id);
917         interface::log_this(log_str);
918     }
919     notice3.settcolor(GREEN);
920     notice3 << "Employee deletion successful!!" << ui::endl;
921     notice3.setexit_button("Back");
922     notice3.loop();
923     notice3.hide();
924 }
925
926 void emp_mgmt::edit_emp()
927 {
928     void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
            among employee, doctor, nurse and receptionist classes
929     if(temp == NULL)
930     {
931         interface::log_this("emp_mgmt::edit_emp() : Not enough memory to allocate
```

```
                                buffer void * temp = malloc( sizeof(doctor) )");
932         interface::error("Out of memory!! Check log");
933         getch();
934         return;
935     }
936     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;
937     unsigned sex_choice;
938     Date dob;
939     address adr;
940     phone phn_no;
941     unsigned long salary, id;
942     Time shift_start, shift_end;
943     const coord menu2_corner_top_left(ui::scr_width * 0.2, 5);
944     const int menu2_width = ui::scr_width * 0.6;
945     menu2:
946     {
947         const int menu2_height = 10;
948         box menu2(menu2_corner_top_left, menu2_width, menu2_height);
949         menu2.settcolor(GREEN);
950         menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                ;
951         menu2.settcolor(WHITE);
952         menu2 << "Edit employee data" << ui::endl;
953         menu2.settcolor(ui::tcolor);
954         menu2 << "Step 1: Enter employee's id: ";
955         menu2.settcolor_input(YELLOW);
956         menu2 >> id;
957         menu2 << ui::endl;
958         menu2.setexit_button("Submit");
959         menu2.setback_func(back_func::set_backbit);
960         menu2.loop();
961         menu2.hide();
962     }
963     if(back_func::backbit)
964     {
965         back_func::backbit = 0;
966         free(temp);
967         return;
968     }
969     if(!hospital::get_employee_by_id(id, temp))
970     {
971         interface::error("ID not found or error while reading from file!");
972         getch();
973         free(temp);
974         return;
975     }
976     notice:
977     {
978         const int notice_height = 14;
979         box notice(menu2_corner_top_left, menu2_width, notice_height);
980         notice.settcolor(GREEN);
981         notice << ui::centeralign << "Employee Management" << ui::endl << ui::
                endl;
982         notice.settcolor(WHITE);
983         notice << "Details of the employee will now be shown with the existing
                data filled. "
984                << "Change the data fields that you require to change, and leave
```

51

```cpp
                          the other data fields as they are. "
985                 << "When you are finished, press Submit to submit the new details.
                          " << ui::endl;
986          notice.settcolor(ui::tcolor);
987          notice << ui::endl;
988          notice.setexit_button("View employee...");
989          notice.setback_func(back_func::set_backbit);
990          notice.loop();
991          notice.hide();
992      }
993      if(back_func::backbit)
994      {
995          back_func::backbit = 0;
996          goto menu2;
997      }
998      employee *e = (employee *) temp;
999      menu3:
1000     {
1001         const int menu3_height = 18;
1002         box menu3( menu2_corner_top_left, menu2_width, menu3_height );
1003         menu3.settcolor(GREEN);
1004         menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                   ;
1005         menu3.settcolor(WHITE);
1006         menu3 << "Edit employee data" << ui::endl;
1007         menu3.settcolor(ui::tcolor);
1008         menu3 << "Step 2: Edit employee details" << ui::endl << ui::endl;
1009
1010         menu3.settcolor_input(YELLOW);
1011         menu3 << "Name: ";
1012         menu3.setdefault( e->get_name() );
1013         menu3 >> name;
1014         menu3 << "Sex: 1. Male | 2. Female | 3. Transsexual" << ui::endl
1015                 << "———— Enter your choice: ";
1016         validate_menu::set_menu_limits(1, 3);
1017         menu3.setdefault( e->get_sex() + 1 );
1018         menu3 >> validate_menu::input >> (int)sex_choice;
1019         menu3 << "Date of Birth(DD/MM/YYYY): ";
1020         sprintf(default_dob_str, "%u/%u/%u", e->get_dob().day, e->get_dob().month
                   , e->get_dob().year);
1021         menu3.setdefault( default_dob_str );
1022         menu3 >> hospital::date_validity >> dob_str;
1023         menu3 << "Address: " << ui::endl;
1024         menu3 << (char)26 << "House no.: ";
1025         menu3.setdefault( e->get_address().house_no );
1026         menu3 >> adr_hno;
1027         menu3 << (char)26 << "Street: ";
1028         menu3.setdefault( e->get_address().street );
1029         menu3 >> adr_street;
1030         menu3 << (char)26 << "City: ";
1031         menu3.setdefault( e->get_address().city );
1032         menu3 >> adr_city;
1033         menu3 << (char)26 << "District: ";
1034         menu3.setdefault( e->get_address().district );
1035         menu3 >> adr_dist;
1036         menu3 << (char)26 << "State: ";
1037         menu3.setdefault( e->get_address().state );
1038         menu3 >> adr_state;
1039         menu3 << "Phone no.: ";
```

```
1040            menu3.setdefault( e->get_phone() );
1041            menu3 >> phn_no;
1042            menu3 << "Salary: ";
1043            menu3.setdefault( e->get_salary() );
1044            menu3 >> salary;
1045            menu3 << "Shift timings: Starts — (HH:MM:SS)";
1046            sprintf(default_shift_str, "%u:%u:%u", e->get_shift(START).hour, e->
                    get_shift(START).minute, e->get_shift(START).second );
1047            menu3.setdefault( default_shift_str );
1048            menu3 >> hospital::time_validity >> shift_start_str;
1049            menu3 << "————————————: Ends — (HH:MM:SS)";
1050            sprintf(default_shift_str, "%u:%u:%u", e->get_shift(END).hour, e->
                    get_shift(END).minute, e->get_shift(END).second );
1051            menu3.setdefault( default_shift_str );
1052            menu3 >> hospital::time_validity >> shift_end_str;
1053            menu3.setexit_button("Submit");
1054            menu3.setback_func(back_func::set_backbit);
1055            menu3.loop();
1056            menu3.hide();
1057        }
1058        if(back_func::backbit)
1059        {
1060            back_func::backbit = 0;
1061            goto notice;
1062        }
1063        ——sex_choice;
1064        dob = hospital::str_to_date(dob_str);
1065        adr = address(adr_hno, adr_street, adr_city, adr_dist, adr_state);
1066        shift_start = hospital::str_to_time(shift_start_str);
1067        shift_end = hospital::str_to_time(shift_end_str);
1068        e->set_name(name);
1069        e->set_sex(sex_choice);
1070        e->set_dob(dob);
1071        e->set_address(adr);
1072        e->set_phone(phn_no);
1073        e->set_salary(salary);
1074        e->set_shift(START, shift_start);
1075        e->set_shift(END, shift_end);
1076        if(id_to_emp::convert(id) == DOCTOR)
1077        {
1078            coord c(1, 4);
1079            doctor *d = (doctor *)temp;
1080            box menu4(c, (ui::scr_width / 2), ui::scr_height — 5);
1081            box inp_box(( c + coord((ui::scr_width / 2) — 1, 0)), (ui::scr_width / 2)
                    + 1, ui::scr_height — 5);
1082            menu4.f << ( ui::top | ui::left ) << (char)204
1083                    << ( ui::bottom | ui::left ) << (char)204
1084                    << ( ui::top | ui::right ) << (char)203
1085                    << ( ui::bottom | ui::right ) << (char)202;
1086            menu4.f.display();
1087            inp_box.f << ( ui::top | ui::left ) << (char)203
1088                << ( ui::bottom | ui::left ) << (char)202
1089                << ( ui::top | ui::right ) << (char)185
1090                << ( ui::bottom | ui::right ) << (char)185;
1091            inp_box.f.display();
1092            menu4 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                    ;
1093            menu4.settcolor(WHITE);
1094            menu4 << "Edit employee data" << ui::endl;
```

```
1095              menu4.settcolor(ui::tcolor);
1096              menu4 << "Step 3: Edit doctor details" << ui::endl << ui::endl;
1097              int speciality[2];
1098              menu4 << "Specialization of doctor (max 2)" << ui::endl
1099                    << "Choose from the following list: " << ui::endl;
1100              for(int i = 0; i <= GEN; ++i)
1101              {
1102                  if(i <= 8)
1103                  {
1104                      menu4 << i << ". " << (body_parts)i << ui::endl;
1105                  }
1106                  else
1107                  {
1108                      inp_box << i << ". " << (body_parts)i << ui::endl;
1109                  }
1110              }
1111              inp_box.settcolor_input(YELLOW);
1112              inp_box << "Enter the number corresponding to the required entry in the 2
                      fields below" << ui::endl;
1113              validate_menu::set_menu_limits(BRAIN, GEN);
1114              inp_box << (char)26;    inp_box.setdefault(d->get_speciality()[0]);
                      inp_box >> validate_menu::input >> speciality[0];
1115              inp_box << (char)26;    inp_box.setdefault(d->get_speciality()[1]);
                      inp_box >> validate_menu::input >> speciality[1];
1116              inp_box << ui::endl;
1117              inp_box.setexit_button("Submit");
1118              inp_box.setback_func(back_func::set_backbit);
1119              inp_box.loop();
1120              menu4.hide();
1121              inp_box.hide();
1122              window.f.display();
1123              d->set_speciality(speciality);
1124          }
1125          if(back_func::backbit)
1126          {
1127              back_func::backbit = 0;
1128              goto menu3;
1129          }
1130          const int notice2_height = 12;
1131          box notice2(menu2_corner_top_left, menu2_width, notice2_height);
1132          notice2.settcolor(GREEN);
1133          notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
1134          if(!hospital::write_employee(temp))
1135          {
1136              notice2.settcolor(RED);
1137              notice2 << "Employee edit unsuccessful!!";
1138              notice2.setexit_button("Exit");
1139              notice2.loop();
1140              notice2.hide();
1141              free(temp);
1142              return;
1143          }
1144          notice2 << "Employee edited successfully!!" << ui::endl;
1145          notice2.settcolor(WHITE);
1146          notice2 << "Hit the button below to display the details you entered: " << ui
                  ::endl;
1147          notice2.settcolor(ui::tcolor);
1148          notice2 << ui::endl;
1149          notice2.setexit_button("View employee...");
```

```
1150        notice2.loop();
1151        notice2.hide();
1152        view_emp(id);
1153        free(temp);
1154    }
1155
1156    void emp_mgmt::pay_emp()
1157    {
1158        unsigned long id;
1159        char ch;
1160        const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
1161        const int menu2_width = ui::scr_width * 0.6;
1162        const int menu2_height = 10;
1163        menu2:
1164        {
1165            box menu2(menu2_corner_top_left, menu2_width, menu2_height);
1166            menu2.settcolor(GREEN);
1167            menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                    ;
1168            menu2.settcolor(WHITE);
1169            menu2 << "Pay salary to individual employee" << ui::endl;
1170            menu2.settcolor(ui::tcolor);
1171            menu2 << "Enter employee's id: ";
1172            menu2.settcolor_input(YELLOW);
1173            menu2 >> id;
1174            menu2 << ui::endl;
1175            menu2.setexit_button("Submit");
1176            menu2.setback_func(back_func::set_backbit);
1177            menu2.loop();
1178            menu2.hide();
1179        }
1180        if(back_func::backbit)
1181        {
1182            back_func::backbit = 0;
1183            return;
1184        }
1185        notice:
1186        {
1187            const int notice_height = 14;
1188            box notice(menu2_corner_top_left, menu2_width, notice_height);
1189            notice.settcolor(GREEN);
1190            notice << ui::centeralign << "Employee Management" << ui::endl << ui::
                    endl;
1191            notice.settcolor(WHITE);
1192            notice << "Hit the button below to display the details of the employee
                    you want to pay salary to: " << ui::endl;
1193            notice.settcolor(ui::tcolor);
1194            notice << ui::endl;
1195            notice.setexit_button("View employee...");
1196            notice.setback_func(back_func::set_backbit);
1197            notice.loop();
1198            notice.hide();
1199        }
1200        if(back_func::backbit)
1201        {
1202            back_func::backbit = 0;
1203            goto menu2;
1204        }
1205        if( !view_emp(id) )
```

```
1206          {
1207              return;
1208          }
1209          {
1210              const int notice2_height = 14;
1211              box notice2( menu2_corner_top_left, menu2_width, notice2_height );
1212              notice2.settcolor(GREEN);
1213              notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::
                        endl;
1214              notice2.settcolor(WHITE);
1215              notice2 << "Are you sure you want to pay salary to this employee?(y/n): "
                        << ui::endl;
1216              notice2.settcolor_input(YELLOW);
1217              notice2 >> ch;
1218              notice2.settcolor(ui::tcolor);
1219              notice2 << ui::endl;
1220              notice2.setexit_button("Submit");
1221              notice2.setback_func(back_func::set_backbit);
1222              notice2.loop();
1223              notice2.hide();
1224          }
1225          if(back_func::backbit)
1226          {
1227              back_func::backbit = 0;
1228              goto notice;
1229          }
1230          if(ch == 'n' || ch == 'N')
1231          {
1232              return;
1233          }
1234          const int notice3_height = 14;
1235          box notice3( menu2_corner_top_left, menu2_width, notice3_height );
1236          notice3.settcolor(GREEN);
1237          notice3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
1238          notice3.settcolor(RED);
1239          if( !hospital::pay_salary(id, system::get_date(), system::get_time()) )
1240          {
1241              notice3 << "Failed to pay salary to the employee! Check log";
1242              notice3.setexit_button("Back");
1243              notice3.loop();
1244              notice3.hide();
1245              return;
1246          }
1247          notice3.settcolor(GREEN);
1248          notice3 << "Pay salary successful!!" << ui::endl;
1249          notice3.setexit_button("Back");
1250          notice3.loop();
1251          notice3.hide();
1252  }
1253
1254  void emp_mgmt::pay_all()
1255  {
1256      char ch;
1257      const int menu2_height = 11;
1258      box menu2(coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, menu2_height);
1259      menu2.settcolor(GREEN);
1260      menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
1261      menu2.settcolor(WHITE);
1262      menu2 << "Pay salary to all employees" << ui::endl;
```

```
1263    menu2.settcolor(ui::tcolor);
1264    menu2 << "Are you sure you want to pay salary to all employees?(y/n): ";
1265    menu2.settcolor_input(YELLOW);
1266    menu2 >> ch;
1267    menu2 << ui::endl;
1268    menu2.setexit_button("Submit");
1269    menu2.loop();
1270    menu2.hide();
1271    if(ch == 'n' || ch == 'N')
1272    {
1273        return;
1274    }
1275    const int notice_height = 10;
1276    box notice( menu2.getcorner_top_left(), menu2.getwidth(), notice_height );
        notice.hide();
1277    box notice2( notice.getcorner_top_left(), notice.getwidth(), notice.getheight
        () );
1278    notice2.settcolor(GREEN);
1279    notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
1280    notice2.hide(); notice.display();
1281    notice.settcolor(GREEN);
1282    notice << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
1283    notice.settcolor(ui::tcolor);
1284    notice << "Pay all salaries in progress..." << ui::endl;
1285    if( !hospital::pay_all_salaries() )
1286    {
1287        notice.hide();
1288        notice2.settcolor(RED);
1289        notice2 << "Failed to pay salary to all employees! Check log";
1290        notice2.setexit_button("Back");
1291        notice2.loop();
1292        notice2.hide();
1293        return;
1294    }
1295    notice.hide();  notice2.display();
1296    notice2 << "Pay all salaries successful!!" << ui::endl;
1297    notice2.setexit_button("Back");
1298    notice2.loop();
1299    notice2.hide();
1300 }
```

2. code/BASE.CPP

```cpp
1  #include "base.hpp"
2
3  ///////////////////////////////////////////////
4  //// Function definitions for class person
5
6  person::person(str inp1, int inp2, Date inp3, address inp4, phone inp5)
7  {
8      strcpy(name, inp1);
9      sex = inp2;
10     dob = inp3;
11     adr = inp4;
12     strcpy(phone_no, inp5);
13     calc_age();
14 }
15
```

```
16  person::person()
17  {
18      strcpy(name, "");
19      dob = Date();
20      strcpy(phone_no, "");
21  }
22
23  char* person::get_name()
24  {
25      return name;
26  }
27
28  int person::get_age()
29  {
30      return age;
31  }
32
33  int person::get_sex()
34  {
35      return sex;
36  }
37
38  Date person::get_dob()
39  {
40      return dob;
41  }
42
43  address person::get_address()
44  {
45      return adr;
46  }
47
48  char* person::get_phone()
49  {
50      return phone_no;
51  }
52
53  void person::calc_age(Date dnow)
54  {
55      if(dnow.month > dob.month || dnow.month == dob.month && dnow.day >= dob.day)
56      {
57          age = dnow.year − dob.year;
58      }
59      else
60      {
61          age = dnow.year − dob.year − 1;
62      }
63  }
64
65  void person::set_name(char* a)
66  {
67      strcpy(name, a);
68  }
69
70  void person::set_sex(int a)
71  {
72      sex = a;
73  }
74
```

```
75   void person::set_dob(Date bday, Date dnow)
76   {
77       dob = bday;
78       calc_age(dnow);
79   }
80
81   void person::set_address(address a)
82   {
83       adr = a;
84   }
85
86   void person::set_phone(char* a)
87   {
88       strcpy(phone_no, a);
89   }
90
91   Time::Time()
92   {
93       hour = 25;
94       minute = 0;
95       second = 0;
96   }
97
98   Time::Time(unsigned h, unsigned m, unsigned s)
99   {
100      hour = h;
101      minute = m;
102      second = s;
103  }
104
105  Date::Date()
106  {
107      day = 0;
108      month = 0;
109      year = 0;
110  };
111
112  Date::Date(unsigned d, unsigned m, unsigned y)
113  {
114      if( d<=31 && m <=12)
115      {
116          day = d;
117          month = m;
118          year = y;
119      }
120      else
121      {
122          day = 0;
123          month = 0;
124          year = 0;
125      }
126  }
127
128  Date system::get_date()
129  {
130      time_t t = time(0);
131      struct tm *now = localtime(&t);
132      Date dnow(now->tm_mday, (now->tm_mon + 1) ,(now->tm_year + 1900));
133      return dnow;
```

59

```
134  }
135
136  Time system::get_time()
137  {
138      time_t t = time(0);
139      struct tm *now = localtime(&t);
140      Time tnow(now->tm_hour, now->tm_min ,now->tm_sec);
141      return tnow;
142  }
143
144  address::address(const char *hno, const char *strt, const char *cty, const char *
         dist, const char *stat)
145  {
146      strcpy(house_no, hno);
147      strcpy(street, strt);
148      strcpy(city, cty);
149      strcpy(district, dist);
150      strcpy(state, stat);
151  }
152
153  userid::userid(str name ,str plaintext) //plaintext is the unencrypted password
154  {
155      strcpy(username, name);
156      set_key(plaintext);
157      makecipher(plaintext);
158  }
159
160  userid::userid()
161  {
162      strcpy(username, "");
163      strcpy(passcipher, "");
164  }
165
166  void userid::makecipher(str plaintext)
167  {
168      int len = strlen(plaintext);
169      int keylen = strlen(default_key);
170      for(int i = 0; i < len; ++i)
171      {
172          int plntext_i = (int)plaintext[i] + 127;
173          int key_i = (int)default_key[i % keylen] + 127;
174          passcipher[i] = (char) ( ( (plntext_i + key_i) % 256 ) - 127);
175      }
176      passcipher[i] = '\0';
177  }
178
179  void userid::set_key(char * plaintext)
180  {
181      randomize();
182      int len = strlen(plaintext);
183      int keylen = random(len/2 + 1) + len/2; //so that the key is not too short
184      for (int i = 0; i <=keylen; ++i)
185      {
186          default_key[i] = (char)( random(256) - 127 );
187      }
188      default_key[i] = '\0';
189  }
190
191  char * userid::decipher()
```

```
192  {
193      str decryptedpass;
194      int len = strlen(passcipher);
195      int keylen = strlen(default_key);
196      for(int i = 0; i < len; ++i)
197      {
198          int cipher_i = (int)passcipher[i] + 127;
199          int key_i = (int)default_key[i % keylen] + 127;
200          decryptedpass[i] = (char) ( ( (cipher_i - key_i + 256) % 256 ) - 127);
201      }
202      decryptedpass[i] = '\0';
203      return decryptedpass;
204  }
205
206  char * userid::get_username()
207  {
208      return username;
209  }
210
211  void userid::set_username(char * inp)
212  {
213      strcpy(username, inp);
214  }
215
216  int userid::login(char * password)
217  {
218      if(!strcmp(password, decipher()))
219          return 1;
220      else
221          return 0;
222  }
223
224  transaction::transaction(float a, Date d, Time t, char * b)
225  {
226      amount = a;
227      strcpy(reason, b);
228      _date = d;
229      _time = t;
230  }
231
232  transaction::transaction()
233  {
234      amount = 0;
235      strcpy(reason, "NA");
236      _date = Date();
237      _time = Time();
238  }
239
240  box & operator<<(box &output, sex s)
241  {
242      switch(s)
243      {
244          case MALE:
245              return output << "Male";
246          case FEMALE:
247              return output << "Female";
248          case TRANS:
249              return output << "Transsexual";
250          default:
```

```cpp
251            return output << "Invalid";
252        }
253 }
254
255 box & operator<<(box &output, body_parts b)
256 {
257     switch(b)
258     {
259         case BRAIN:
260             return output << "Brain";
261         case HEART:
262             return output << "Heart";
263         case SKIN:
264             return output << "Skin";
265         case LUNG:
266             return output << "Lung";
267         case BONE:
268             return output << "Bone";
269         case EYE:
270             return output << "Eye";
271         case THROAT:
272             return output << "Throat";
273        case TEETH:
274             return output << "Teeth";
275        case STOMACH:
276             return output << "Stomach";
277        case BLOOD:
278             return output << "Blood";
279        case GUT:
280             return output << "Gastrointestinal tract";
281        case GEN:
282             return output << "General ailments";
283        default:
284             return output << "Invalid";
285    }
286 }
287
288 box & operator<<(box &output, Time & t)
289 {
290     return output << (unsigned long)t.hour << ':' << (unsigned long)t.minute << '
            :' << (unsigned long)t.second;
291 }
292
293 box & operator<<(box &output, Date & d)
294 {
295     return output << (unsigned long)d.day << '/' << (unsigned long)d.month << '/'
            << (unsigned long)d.year;
296 }
297
298 box & operator<<(box &output, address & a)
299 {
300     return output << a.house_no << ", " << a.street << ", "
301             << a.city << ", " << a.district << ", " << a.state;
302 }
```

3. code/HOSP.CPP

```cpp
1 #include "hosp.hpp"
```

```cpp
2   #include "iface.hpp"
3   #include "emp.hpp"
4   #include <fstream.h>
5
6   /////////////////////////////////////////////////
7   ///////////////////////////////////////////////// Function definitions for class
        hospital
8
9   float hospital::get_balance(){
10      return balance;
11  }
12
13  transaction hospital::deduct_money(float amt, char* reason, Date d, Time t){
14      hospital::balance -= amt;
15
16      ofstream hosp_finances ("transactions.dat", ios::out | ios::binary | ios::app
            );
17
18      transaction temp = transaction( (-1)*amt, d, t, reason);
19
20      hosp_finances.write( (char*) (&temp) , sizeof(transaction) );
21
22      hosp_finances.close();
23
24      return temp;
25  }
26
27  transaction hospital::add_money(float amt, char* reason, Date d, Time t){
28      hospital::balance += amt;
29
30      ofstream hosp_finances ("transactions.dat", ios::out | ios::binary | ios::app
            );
31
32      transaction temp = transaction( (-1)*amt,d, t, reason);
33
34      hosp_finances.write( (char*) (&temp) , sizeof(transaction) );
35
36      hosp_finances.close();
37
38      return temp;
39  }
40
41  transaction* hospital::get_transaction(){
42      transaction a[10];
43
44      ifstream hosp_finances ("transactions.dat", ios::in | ios::binary);
45
46      hosp_finances.seekg( (-1) * sizeof(transaction) , hosp_finances.end );
47
48      for(int i = 0; i < 10; i++){
49          hosp_finances.read( (char *) &a[i] , sizeof(transaction) );
50          hosp_finances.seekg( hosp_finances.tellg() - ( 2 * sizeof(transaction) )
                );
51      }
52
53      return a;
54  }
55
56  patient hospital::get_patient_by_id(long id){
```

```
57        patient a;

58

59        str temp;

60

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

62

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

64

65        if(!i){
66            interface::error("File read error!!");
67            getch();
68        }

69

70        return a;

71

72   }

73

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

81

82        if(patient_file){
83            patient_file.write( (char*) &a , sizeof(patient) );
84        }
85        else{
86            interface::error("Patient file access failure!!");
87        }
88        if(patient_file.fail()){
89            interface::error("Patient file write failure!!");
90        }
91        patient_file.close();
92   }

93

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

96

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

102

103       hospital::write_patient(temp_pat);
104  }

105

106  void hospital::discharge_patient(patient temp){
107       temp.discharge();
108       temp.set_discharge_date( system::get_date() );
109       hospital::write_patient(temp);
110  }

111

112  float hospital::calc_bill(int stay){
113       return stay * ::stay_charge;
114  }

115
```

```
116  medicine hospital::get_med_by_code(int inp_code){
117      fstream meds ("stock/med.dat", ios::in | ios::binary);
118
119      medicine temp;
120
121      if(inp_code < 1 || inp_code > 100){
122          temp.code = 0;
123          temp.price = 0;
124          temp.dosage = 0;
125          temp.stock = 0;
126          strcpy(temp.name, "Shell Medicine");
127
128          interface::error("Invalid medicine code!!");
129
130          return temp;
131      }
132
133      for(int i = 0; i<100; i++){
134          meds.read((char*) &temp, sizeof(medicine));
135          if(temp.code == inp_code){
136              break;
137          }
138      }
139
140      return temp;
141  }
142
143  void hospital::write_med(medicine inp_med){
144      fstream med_file ("stock/med.dat", ios::in | ios::out | ios::binary);
145      med_file.seekg(0);
146
147      int success = 0;
148
149      while (!success){
150          medicine a;
151          med_file.read( (char*) &a, sizeof(medicine) );
152          if(a.code==inp_med.code){
153              med_file.seekg( med_file.tellg() − sizeof(medicine) );
154              med_file.write( (char*) &a, sizeof(medicine) );
155              success++;
156          }
157      }
158
159  }
160
161  int hospital::get_employee_by_id(unsigned long ID, void * target)
162  {
163      if(target == NULL)
164      {
165          interface::log_this("hospital::get_employee_by_id() : NULL pointer
                  supplied to function\nFunction aborted");
166          return 0;
167      }
168      str temp;
169      int size_of_target;
170      switch(id_to_emp::convert(ID))
171      {
172          case INVALID:
173              interface::log_this("hospital::get_employee_by_id() : Invalid id
```

```
                             supplied to function\nFunction aborted");
174             return 0;
175         case OTHERS:
176             sprintf(temp, "employee/%lu/base.dat", ID);
177             size_of_target = sizeof(employee);
178             break;
179         case DOCTOR:
180             sprintf(temp, "employee/doctor/%lu/base.dat", ID);
181             size_of_target = sizeof(doctor);
182             break;
183         case NURSE:
184             sprintf(temp, "employee/nurse/%lu/base.dat", ID);
185             size_of_target = sizeof(nurse);
186             break;
187         case RECEPTIONIST:
188             sprintf(temp, "employee/receptionist/%lu/base.dat", ID);
189             size_of_target = sizeof(receptionist);
190             break;
191     }
192     int i = hospital::read_from( ID, (char*) target, size_of_target, temp );
193     if(!i)
194     {
195         target = NULL;
196         return 0;
197     }
198     return 1;
199 }
200
201 int hospital::write_employee(void * a)
202 {
203     if(a == NULL)
204     {
205         interface::log_this("hospital::write_employee() : NULL pointer supplied
                to function\nFunction aborted");
206         return 0;
207     }
208     mkdir("employee");
209     str temp;
210     int size_of_target;
211     employee *x = (employee *) a;
212     const unsigned long ID = x->get_id();
213     switch(id_to_emp::convert(ID))
214     {
215         case INVALID:
216             interface::log_this("hospital::write_employee() : Object with ID zero
                    cannot be written to file\nFunction aborted");
217             return 0;
218         case OTHERS:
219             sprintf(temp, "employee/%lu", ID);
220             size_of_target = sizeof(employee);
221             break;
222         case DOCTOR:
223             mkdir("employee/doctor");
224             sprintf(temp, "employee/doctor/%lu", ID);
225             size_of_target = sizeof(doctor);
226             break;
227         case NURSE:
228             mkdir("employee/nurse");
229             sprintf(temp, "employee/nurse/%lu", ID);
```

```
230              size_of_target = sizeof(nurse);
231              break;
232          case RECEPTIONIST:
233              mkdir("employee/receptionist");
234              sprintf(temp, "employee/receptionist/%lu", ID);
235              size_of_target = sizeof(receptionist);
236              break;
237      }
238      mkdir(temp);
239      strcat(temp, "/base.dat");
240      ofstream fout ( temp , ios::out | ios::binary);
241      if(!fout)
242      {
243          interface::log_this("hospital::write_employee() : Employee data file
                  could not be created or accessed\nFunction aborted");
244          return 0;
245      }
246      fout.write( (char *) a , size_of_target );
247      if(fout.fail())
248      {
249          interface::log_this("hospital::write_employee() : Error while writing to
                  file (fout.fail())\nFunction aborted");
250          return 0;
251      }
252      return 1;
253  }
254
255  int hospital::pay_salary(unsigned long id, Date d1, Time t1)
256  {
257      void * e = malloc( sizeof(doctor) );
258      if(e == NULL)
259      {
260          interface::log_this("hospital::pay_salary() : Not enough memory to
                  allocate buffer void * temp = malloc( sizeof(doctor) )");
261          interface::error("Out of memory!! Check log");
262          getch();
263          return 0;
264      }
265      str temp;
266      switch(id_to_emp::convert(id))
267      {
268          case INVALID:
269              interface::log_this("hospital::pay_salary() : Invalid id supplied to
                      function\nFunction aborted");
270              return 0;
271          case OTHERS:
272              sprintf(temp, "employee/%lu/trans.dat", id);
273              break;
274          case DOCTOR:
275              sprintf(temp, "employee/doctor/%lu/trans.dat", id);
276              break;
277          case NURSE:
278              sprintf(temp, "employee/nurse/%lu/trans.dat", id);
279              break;
280          case RECEPTIONIST:
281              sprintf(temp, "employee/receptionist/%lu/trans.dat", id);
282              break;
283      }
284      if(!hospital::get_employee_by_id(id, e))
```

```
285        {
286            interface::log_this("hospital::pay_salary() : Employee not found or error
                   while reading file\nFunction aborted");
287            free(e);
288            return 0;
289        }
290        unsigned long inp1;
291        char inp2[100] = "Salary paid to ";
292        employee * emp = (employee *)e;
293        inp1 = emp->get_salary();
294        strcat(inp2, emp->get_name());
295        transaction t = hospital::deduct_money(inp1, inp2, d1, t1);
296        free(e);
297
298        ofstream fout ( temp ,ios::binary | ios::app );
299        if(!fout)
300        {
301            interface::log_this("hospital::pay_salary() : Employee data file could
                   not be created or accessed\nFunction aborted");
302            return 0;
303        }
304        fout.write((char *) &t, sizeof(transaction));
305        if(fout.fail())
306        {
307            interface::log_this("hospital::pay_salary() : Error while writing to file
                   (fout.fail())\nFunction aborted");
308            return 0;
309        }
310        return 1;
311   }
312
313   int hospital::pay_all_salaries()
314   {
315        Date d1 = system::get_date();
316        Time t1 = system::get_time();
317        unsigned long max_id;
318        ifstream fin;
319        fin.open("employee/max_id.dat", ios::binary);
320        if(!fin)
321        {
322            interface::log_this("hospital::pay_all_salaries() : No employees found or
                   cannot access file max_id.dat\nFunction aborted");
323            return 0;
324        }
325        else
326        {
327            fin.read((char *) &max_id, sizeof(unsigned long));
328            if(fin.fail())
329            {
330                interface::log_this("hospital::pay_all_salaries() : Error while
                       reading file max_id.dat(fin.fail())\nFunction aborted");
331                return 0;
332            }
333            if(!employee::get_generate_id_status())
334            {   //if generate_id_status is zero, then no id is generated after max_id
                   + 1
335                //Thus, the following loop should run max_id + 1 times
336                ++max_id;
337            }
```

68

```
338          for(unsigned long i = 1; i <= max_id; ++i)
339          {
340              int a = hospital::pay_salary(i, d1, t1);
341              if(!a)
342              {
343                  str log_msg;
344                  sprintf(log_msg, "hospital::pay_all_salaries() : Failed to pay
                         salary of id %lu...\nSkipped", i);
345                  interface::log_this(log_msg);
346              }
347          }
348      }
349      return 1;
350  }
351
352  int hospital::get_date_difference(Date dt1, Date dt2)
353  {
354
355      long int n1 = dt1.year*365 + dt1.day;
356
357      for (int i=0; i<dt1.month - 1; i++){
358          n1 += monthDays[i];
359      }
360      n1 += hospital::count_leap_years(dt1);
361
362      long int n2 = dt2.year*365 + dt2.day;
363
364      for (i=0; i<dt2.month - 1; i++){
365              n2 += monthDays[i];
366      }
367      n2 += count_leap_years(dt2);
368
369      return (n2 - n1);
370  }
371
372  int hospital::count_leap_years(Date d)
373  {
374      int years = d.year;
375
376      if (d.month <= 2){
377          years--;          // checking whether to count the current year
378      }
379
380      return (years / 4) - (years / 100) + (years / 400);
381  }
382
383  int hospital::date_validity(const char * inp_date){
384      return hospital::date_validity(hospital::str_to_date(inp_date));
385  }
386
387  int hospital::date_validity(Date inp_date){
388       if(inp_date.year % 4 == 0 && inp_date.month == 2 &&
389          inp_date.day == 29){
390              return 1;
391      }
392      if (
393          inp_date.month > 12 ||
394          inp_date.day > monthDays[inp_date.month - 1])
395      {
```

```cpp
396         return 0;
397     }
398     else{
399         return 1;
400     }
401 }
402
403 int hospital::time_validity(const char * inp_time)
404 {
405     return time_validity( str_to_time(inp_time) );
406 }
407
408 int hospital::time_validity(Time t)
409 {
410     if( t.hour > 24 || t.minute > 59 || t.second > 59)
411     {
412         return 0;
413     }
414     return 1;
415 }
416
417 Date hospital::str_to_date(const char * inp_date){
418     int counter = 0;
419     int count = 0;
420     int input[3];
421     input[0] = input[1] = input[2] = 0;
422     while(counter < 3){
423         char ch[12];
424         ch[0] = '/';
425         for(int i = 1; i < 7; i++){
426             ch[i] = inp_date[count];
427             count++;
428             if(ch[i] == '/' || ch[i] == '\\' || ch[i] == 0 || ch[i] == '-'){
429                 if(ch[i] == 0 && count < 11){
430                     interface::error("Invalid date!");
431                     return Date (99, 99, 9999);
432                 }
433                 ch[i] = '/';
434                 int temp = i-1, temp2 = 0;
435                 while(ch[temp] != '/'){
436                     input[counter] += ( pow(10, temp2) * ((int)ch[temp] - (int)'0
437                         ') );
                    temp--;
438                     temp2++;
439                 }
440                 counter++;
441             }
442         }
443     }
444
445     return Date(input[0], input[1], input[2]);
446 }
447
448 Time hospital::str_to_time(const char * inp_time)
449 {
450 ///////In this function invalid time(25:00:00) is returned if time is in
        incorrect format//////////
451     char inp[3][3] = {"25", "0", "0"};
452     int inp_x = 0, inp_y = 0;
```

```cpp
453        Time null(25, 0, 0);
454        if( strlen(inp_time) > 8 || strlen(inp_time) < 5 || inp_time[strlen(inp_time)
               - 1] == ':')
455        {
456            return null;
457        }
458        for(int i = 0; i < strlen(inp_time); ++i)
459        {
460            if(inp_time[i] == ':' && inp_y != 0)
461            {
462                inp[inp_x][inp_y] = '\0';
463                ++inp_x;
464                inp_y = 0;
465                continue;
466            }
467            else if( (inp_y == 0 && inp_time[i] == ':') || inp_y > 1
468                     || (inp_time[i] < '0' || inp_time[i] > '9') )
469            {
470                return null;
471            }
472            inp[inp_x][inp_y] = inp_time[i];
473            ++inp_y;
474        }
475        char *endptr;
476        null.hour = (unsigned int) strtol(inp[0], &endptr, 10);
477        null.minute = (unsigned int) strtol(inp[1], &endptr, 10);
478        null.second = (unsigned int) strtol(inp[2], &endptr, 10);
479        return null;
480  }
481
482  int hospital::str_to_sex(char* s){
483        if( strcmp(s, "M")  )    { return 0; }
484        else if(    strcmp(s, "F")  )    { return 1; }
485        else { return 2; }
486  }
487
488  int hospital::read_from(unsigned long ID, char * dest, int size, char * temp)
489  {
490        ifstream fin ( temp , ios::in | ios::binary );
491        if(!fin)
492        {
493            char errmsg[200];
494            sprintf(errmsg, "hospital::read_from() : Employee with id %lu not found\
                   nFunction aborted", ID);
495            interface::log_this(errmsg);
496            return 0;
497        }
498        fin.read( dest, size );
499        if(fin.fail())
500        {
501            interface::log_this("hospital::read_from() : Error while reading from
                   file (fin.fail())\nFunction aborted");
502            return 0;
503        }
504        fin.close();
505        return 1;
506  }
507
508  ////////////////////////////////////////////////////////////
```

71

```
509   /////////////////////////////////////////////////////////
510   /////////////////////////////////////////////////////////
511
512   double hospital::balance = 10000000.0;
```

4. code/MIAN.CPP

```
1   #include "iface.hpp"
2   #include <conio.h>
3   #include "hosp.hpp"
4   #include "emp.hpp"
5
6   void main()
7   {
8       clrscr();
9   /*///////////////Administrator object creator///////////
10      address yay("", "", "", "", "");
11      employee x("Administrator", 3, Date(), yay, "", 0, Time(), Time(), "admin", "
            password");
12      hospital::write_employee(&x);
13  ////////////////////////////////////////////////////*/
14
15      interface::log_this("Program initiated\n\n");
16
17      interface::init();
18
19      interface::log_this("Program terminated\n\n");
20  }
```

5. code/iface.cpp

```
1   #include <fstream.h>
2   #include "base.hpp"
3   #include "iface.hpp"
4   #include "hosp.hpp"
5   #include "emp.hpp"
6
7   //////////////////////////////////////////////
8   ///////// Function definitions for interface
9
10  void interface::stock_management(){
11      coord c(ui::scr_width / 3, ui::scr_height / 3);
12      box menu (c, ui::scr_width / 3, ui::scr_height / 2.2);
13
14      int ch = 0;
15
16      menu << "1. Sale"
17              << ui::endl << "2. Purchase"
18              << ui::endl << "3. Stock check"
19              << ui::endl << "4. Go to main menu"
20              << ui::endl << ui::endl << "Choice : ";
21      menu.setdefault(1);
22      menu.settcolor_input(YELLOW);
23      validate_menu::set_menu_limits(1, 4);
24      menu >> validate_menu::input >> ch;
25
26      menu << ui::endl;
```

```cpp
27          menu.setexit_button("Submit");

28
29          menu.loop();
30          menu.hide();

31
32          interface::clear_error();

33
34          switch(ch){
35              case 1:
36              {

37
38                  medicine temp;
39                  temp.code = 0;

40
41                  while(temp.code == 0){
42                      coord c(ui::scr_width / 3, ui::scr_height / 3);
43                      box sale_menu (c, ui::scr_width / 3, ui::scr_height / 3);
44                      sale_menu.settcolor_input(YELLOW);
45                      sale_menu << ui::centeralign << "Medicine Sale" << ui::endl;
46                      sale_menu << "Code : ";
47                      sale_menu.setdefault(42);
48                      sale_menu >> temp.code;
49                      sale_menu << ui::endl;
50                      sale_menu.setexit_button("Submit");
51                      sale_menu.loop();
52                      sale_menu.hide();

53
54                      temp = hospital::get_med_by_code(temp.code);
55                  }

56
57                  int quantity = -2;
58                  patient temp_patient;
59                  long pat_id;

60
61                  while(quantity < 0 || quantity > 100){
62                      coord c(ui::scr_width / 3, ui::scr_height / 3);
63                      box sale_menu (c, ui::scr_width / 3, ui::scr_height / 2);
64                      sale_menu.settcolor_input(YELLOW);
65                      sale_menu << ui::centeralign << "Medicine Sale" << ui::endl;
66                      sale_menu << "Name : " << temp.name
67                                << ui::endl << "Price : $" << temp.price
68                                << ui::endl << ui::endl
69                                << "Patient ID : ";
70                      sale_menu.setdefault(786);
71                      sale_menu >> pat_id;
72                      sale_menu << ui::endl << "Quantity : ";
73                      sale_menu.setdefault(1);
74                      sale_menu >> quantity;
75                      sale_menu.setexit_button("Submit");
76                      sale_menu.loop();
77                      sale_menu.hide();

78
79                      temp_patient = hospital::get_patient_by_id(pat_id);
80                      if(temp_patient.get_id() == 0){
81                          quantity = -1;
82                          interface::error("Invalid patient ID!!");
83                          continue;
84                      }
85                      interface::error("Invalid quantity!!");
```

73

```
86                  }
87
88                  interface::clear_error();
89
90                  temp.stock -= quantity;
91
92                  for(int i = 0; i < 50; i++){
93                      if(temp_patient.get_med(i, 0) == temp.code ||
94                          temp_patient.get_med(i,0) == 0){
95                              temp_patient.set_med(i, temp.code, temp_patient.
                                    get_med(i, 1) + quantity);
96                          }
97                  }
98
99                  hospital::write_patient(temp_patient);
100                 hospital::write_med(temp);
101
102                 break;
103             }
104
105         case 2:
106         {
107             medicine temp;
108             temp.code = 0;
109
110             while(temp.code == 0){
111                 coord c(ui::scr_width / 3, ui::scr_height / 3);
112                 box purchase_menu (c, ui::scr_width / 3, ui::scr_height / 3);
113                 purchase_menu.settcolor_input(YELLOW);
114                 purchase_menu << ui::centeralign << "Medicine Purchase" << ui::
                        endl;
115                 purchase_menu << "Code : ";
116                 purchase_menu.setdefault(42);
117                 purchase_menu >> temp.code;
118                 purchase_menu << ui::endl;
119                 purchase_menu.setexit_button("Submit");
120                 purchase_menu.loop();
121                 purchase_menu.hide();
122
123                 temp = hospital::get_med_by_code(temp.code);
124             }
125
126             int quantity = -2;
127
128             while(quantity < 0 || quantity > 5000){
129                 coord c(ui::scr_width / 3, ui::scr_height / 3);
130                 box purchase_menu (c, ui::scr_width / 3, ui::scr_height / 2);
131                 purchase_menu.settcolor_input(YELLOW);
132                 purchase_menu << ui::centeralign << "Medicine Purchase" << ui::
                        endl;
133                 purchase_menu << "Name : " << temp.name
134                             << ui::endl << "Price : $" << temp.price
135                             << ui::endl << ui::endl << "Quantity : ";
136                 purchase_menu.setdefault(1);
137                 purchase_menu >> quantity;
138                 purchase_menu.setexit_button("Submit");
139                 purchase_menu.loop();
140                 purchase_menu.hide();
141
```

```cpp
142                    interface::error("Invalid quantity!!");
143                }
144
145                interface::clear_error();
146
147                temp.stock += quantity;
148                hospital::deduct_money(temp.price * quantity, "Medicine purchase",
                        system::get_date(), system::get_time());
149                hospital::write_med(temp);
150
151                break;
152            }
153
154            case 3:
155            {
156                medicine temp;
157                temp.code = 0;
158
159                while(temp.code == 0){
160                    coord c(ui::scr_width / 3, ui::scr_height / 3);
161                    box stock_menu (c, ui::scr_width / 3, ui::scr_height / 3);
162                    stock_menu.settcolor_input(YELLOW);
163                    stock_menu << ui::centeralign << "Stock check" << ui::endl;
164                    stock_menu << "Code : ";
165                    stock_menu.setdefault(42);
166                    stock_menu >> temp.code;
167                    stock_menu << ui::endl;
168                    stock_menu.setexit_button("Submit");
169                    stock_menu.loop();
170                    stock_menu.hide();
171
172                    temp = hospital::get_med_by_code(temp.code);
173                }
174
175                coord c(ui::scr_width / 3, ui::scr_height / 3);
176                box stock_menu (c, ui::scr_width / 3, ui::scr_height / 2);
177                stock_menu.settcolor_input(YELLOW);
178                stock_menu << ui::centeralign << "Medicine Details" << ui::endl;
179                stock_menu << "Name : " << temp.name
180                            << ui::endl << "Price : $" << temp.price
181                            << ui::endl << "Dosage : " << temp.dosage << " ml"
182                            << ui::endl << "Quantity in stock : " << temp.stock
183                            << ui::endl;
184                stock_menu.setexit_button("Okay");
185                stock_menu.loop();
186                stock_menu.hide();
187
188                break;
189            }
190        }
191
192 }
193
194 int interface::validate_menu::input(const char * ch)
195 {
196     char *endptr;
197     int a = (int) strtol(ch, &endptr, 10);
198     if(!validation::vint(ch) || a < lowest_choice || a > greatest_choice)
199     {
```

```
200          return 0;
201      }
202      else
203      {
204          return 1;
205      }
206 }
207
208 void interface::validate_menu::set_menu_limits(int a, int b)
209 {
210      lowest_choice = a;
211      greatest_choice = b;
212 }
213
214 int interface::validate_menu::lowest_choice = 0;
215 int interface::validate_menu::greatest_choice = 0;
216
217 int interface::back_func::set_backbit()
218 {
219      backbit = 1;
220      return 1;
221 }
222
223 int interface::back_func::backbit = 0;
224
225 void interface::error(char* err){
226      window.clear_footer();
227      window.setfooter_tcolor(RED);
228      window << box::setfooter << ui::centeralign
229          << err;
230 }
231
232 void interface::clear_error(){
233      window.clear_footer();
234      window.setfooter_tcolor(GREEN);
235      window << box::setfooter << ui::centeralign
236          << "Everything looks OK";
237 }
238
239 int interface::log_this(char * message)
240 {
241      Date dnow = system::get_date();
242      Time tnow = system::get_time();
243      char text[300];
244      sprintf(text, "$ [%u-%u-%u %u:%u:%u +0530]: ", dnow.day, dnow.month, dnow.
             year, tnow.hour, tnow.minute, tnow.second);
245      strcat(text, message);
246      ofstream fout;
247      fout.open("log.txt", ios::out | ios::app);
248      if(!fout)
249          return 0;
250      fout << text << endl;
251      if(fout.fail())
252          return 0;
253      fout.close();
254      return 1;
255 }
256
257 interface::interface(){}
```

```
258
259  box interface::window;
```

6. code/iface2.cpp

```cpp
1   #include <fstream.h>
2   #include "base.hpp"
3   #include "iface.hpp"
4   #include "hosp.hpp"
5   #include "emp.hpp"
6
7   void interface::init(){
8       window.hide();
9       window.display();
10      window.settcolor(WHITE);
11      window << ui::centeralign << "LHOSPITAL";
12      window.settcolor(ui::tcolor);
13      window.setfooter_tcolor(GREEN);
14
15      Date current_date = system::get_date();
16      Time current_time = system::get_time();
17
18      str curr_date, curr_time;
19      sprintf(curr_date, "%d/%d/%d", current_date.day, current_date.month,
            current_date.year);
20      sprintf(curr_time, "%d:%d", current_time.hour, current_time.minute);
21
22      window << box::setheader << curr_date << box::setheader << ui::rightalign
23              << curr_time << box::setfooter << ui::centeralign
24              << "Everything looks OK";
25      int id;
26      do
27      {
28          id = interface::login_screen();
29          if(id && id_to_emp::convert(id) != OTHERS || id == 1)   //so that general
                employees (except administrator) do
30          {                                                       // not
                accidentally login(as they have been assigned
31              interface::clear_error();                           // username and
                password as "", "")
32              break;
33          }
34      }while(1);
35      if(id == 1) //if user logging in is administrator
36      {
37          int choice = 0;
38
39          while(1){
40              choice = interface::menu();
41
42              switch(choice){
43                  case 1:
44                      interface::employee_management();
45                      break;
46                  case 2:
47                      interface::stock_management();
48                      break;
49                  case 3:
```

77

```
50                          return;
51                      }
52              }
53          }
54          else
55          {
56              switch(id_to_emp::convert(id))
57              {
58                  case INVALID:
59                      interface::error("You have an invalid id generated. Create a new
                             account");
60                      break;
61                  case DOCTOR:
62                  case NURSE:
63                  case RECEPTIONIST:
64                      interface::employee_screen(id);
65                      break;
66              }
67          }
68  }
69
70  int interface::login_screen()
71  {
72      const int login_screen_height = 9;
73      coord c(ui::scr_width / 3, ui::scr_height / 3);
74      box login_box (c, ui::scr_width / 3, login_screen_height);
75
76      str uid, pwd;
77
78      login_box.settcolor_input(YELLOW);
79      login_box << "User ID : ";
80      login_box >> uid;
81      login_box << ui::endl << "Password : ";
82      login_box >> box::setpassword >> pwd;
83      login_box << ui::endl;
84      login_box.setexit_button("Login");
85      login_box.loop();
86      login_box.hide();
87      unsigned long max_id;
88      ifstream fin;
89      fin.open("employee/max_id.dat", ios::binary);
90      if(!fin)
91          max_id = 1;
92      else
93      {
94          fin.read((char *) &max_id, sizeof(unsigned long));
95          if(fin.fail())
96          {
97              interface::error("ERROR WHILE READING FROM FILE!!! ");
98              getch();
99              return 0;
100         }
101     }
102     fin.close();
103     void * x = malloc( sizeof(doctor) );
104     for(unsigned long id = 1; id <= max_id; ++id)
105     {
106         if(x == NULL)
107         {
```

```cpp
108              interface::log_this("interface::login_screen() : Not enough memory to
                     allocate buffer void * temp = malloc( sizeof(doctor) )");
109              interface::error("Out of memory!! Check log");
110              getch();
111              return 0;
112          }
113          if(!hospital::get_employee_by_id(id, x))
114          {
115              char log_msg[300];
116              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);
117              interface::log_this(log_msg);
118          }
119          employee * e = (employee *)x;
120          if(!strcmp(e->account.get_username(), uid) && e->account.login(pwd))
121          {
122              interface::clear_error();
123              free(x);
124              return id;
125          }
126      }
127      interface::error("Invalid login details!!");
128      free(x);
129      return 0;
130  }
131
132  int interface::menu(){
133      coord c(ui::scr_width / 3, ui::scr_height / 3);
134      box menu (c, ui::scr_width / 3, ui::scr_height / 2.2 + 1);
135
136      int ch;
137      menu << ui::endl << "1. Employee management"
138          << ui::endl << "2. Stock management"
139          << ui::endl << "3. Exit"
140          << ui::endl << ui::endl << "Choice : ";
141      menu.settcolor_input(YELLOW);
142      validate_menu::set_menu_limits(1, 3);
143      menu >> validate_menu::input >> ch;
144
145      menu << ui::endl;
146      menu.setexit_button("Submit");
147
148      menu.loop();
149      menu.hide();
150
151      return ch;
152  }
153
154  void interface::patient_management(){
155      int ch = 0;
156
157      coord c(ui::scr_width / 3, ui::scr_height / 3);
158      box menu (c, ui::scr_width / 3, ui::scr_height / 2.2);
159
160      menu << "1. Patient admission"
161          << ui::endl << "2. Patient discharge"
162          << ui::endl << "3. Edit patient details"
163          << ui::endl << "4. Go to main menu"
```

79

```
164                     << ui::endl << ui::endl << "Choice : ";
165         menu.setdefault(1);
166         menu.settcolor_input(YELLOW);
167         validate_menu::set_menu_limits(1,4);
168         menu >> validate_menu::input >> ch;
169
170             menu << ui::endl;
171             menu.setexit_button("Submit");
172
173             menu.loop();
174             menu.hide();
175
176         switch(ch){
177             case 1:
178             {
179                 coord c(ui::scr_width / 4, ui::scr_height / 4);
180                 box form (c, ui::scr_width / 2, ui::scr_height / 1.5);
181                 form.settcolor_input(YELLOW);
182
183                 str inp_name, inp_sex_str, inp_dob_str
184                     , inp_phone, inp_guard_name, inp_emer_contact
185                     , inp_emer_phone, inp_insur_expiry, inp_admdate_str;
186
187                 address inp_adr;
188                 disease inp_dis;
189                 insurance inp_insur;
190
191                 form << "Enter data for the patient :" << ui::endl
192                         << ui::endl << "Name : ";
193                 form >> inp_name;
194
195                 form << ui::endl << "Sex : ";
196                 form >> inp_sex_str;
197                 form << ui::endl << "Key - M/F/T = Male/Female/Trans"
198                         << ui::endl << "Date of Birth : ";
199
200                 form.setdefault("25/12/1991");
201                 form >> inp_dob_str;
202
203
204                 form << ui::endl << "Address"
205                         << ui::endl << ui::endl
206                         << "\tHouse # : ";
207                 form.setdefault("221B");
208                 form >> inp_adr.house_no;
209
210                 form << ui::endl << "\tStreet : ";
211                 form.setdefault("Baker Street");
212                 form >> inp_adr.street;
213
214                 form << ui::endl << "\tDistrict : ";
215                 form.setdefault("Idk");
216                 form >> inp_adr.district;
217
218                 form << ui::endl << "\tState : ";
219                 form.setdefault("London(?)");
220                 form >> inp_adr.state;
221
222
```

```
223            form << ui::endl << ui::endl
224                << "Phone : ";
225            form.setdefault("1234567890");
226            form >> inp_phone;
227
228
229            form << ui::endl << "Disease"
230                << ui::endl << ui::endl
231                << "\tName : ";
232            form.setdefault("Melanoma");
233            form >> inp_dis.name;
234
235            form << ui::endl << "Type : ";
236            form.setdefault(0);
237            form >> inp_dis.type;
238
239            form << ui::endl << "\tType key : " << ui::endl
240                << "\t0 - Brain\t1 - Heart" << ui::endl
241                << "\t2 - Skin\t3 - Lung" << ui::endl
242                << "\t4 - Bone\t5 - Eye" << ui::endl
243                << "\t6 - Throat\t7 - Teeth" << ui::endl
244                << "\t8 - Stomach\t9 - Blood" << ui::endl
245                << "\t10 - General/full body condition"
246                << ui::endl << "\tSymptoms"
247                << ui::endl << "\tSymptom 1 : ";
248
249            form >> inp_dis.symptoms[0];
250
251            form << ui::endl << "\tSymptom 2 : ";
252            form >> inp_dis.symptoms[1];
253
254            form << ui::endl << "\tSymptom 3 : ";
255            form >> inp_dis.symptoms[2];
256
257            form << ui::endl << "\tSymptom 4 : ";
258            form >> inp_dis.symptoms[3];
259
260
261            form << ui::endl << ui::endl
262                << "Guardian name : ";
263            form.setdefault("Dr. John Watson");
264            form >> inp_guard_name;
265
266            form << ui::endl << "Emergency Contact : ";
267            form.setdefault("Irene Adler");
268            form >> inp_emer_contact;
269
270            form << ui::endl << "Emer. Cont. Phone : ";
271            form.setdefault("1234567890");
272            form >> inp_emer_phone;
273
274
275            form << ui::endl << "Insurance"
276                << ui::endl << ui::endl
277                << "\tProvider : ";
278            form.setdefault("LIC");
279            form >> inp_insur.provider;
280
281            form << ui::endl << "\tAmount ($) : ";
```

```
282          form.setdefault(30000);
283          form >> inp_insur.amount;
284
285          form << ui::endl << "\tExpiry";
286          form.setdefault("25/12/2022");
287          form >> inp_insur_expiry;
288
289
290          form << ui::endl << ui::endl
291                  << "Admission Date : ";
292          char dnow[11];
293          form.setdefault("01/01/2018");
294          form >> inp_admdate_str;
295
296          form << ui::endl << ui::endl;
297          form.setexit_button("Submit");
298
299          form.loop();
300
301          form.hide();
302
303          inp_insur.expiry = hospital::str_to_date(inp_insur_expiry);
304
305          patient temp_pat = patient(inp_name, hospital::str_to_sex(inp_sex_str
                  )
306                                  , hospital::str_to_date(inp_dob_str),
                                      inp_adr
307                                  , inp_phone, inp_dis, inp_guard_name
308                                  , inp_emer_contact, inp_emer_phone
309                                  , inp_insur, hospital::str_to_date(
                                      inp_admdate_str));
310
311          hospital::write_patient(temp_pat);
312
313          coord d(ui::scr_width / 3, ui::scr_height / 3);
314          box message (d, ui::scr_width / 3, ui::scr_height / 3);
315
316          message << ui::centeralign << "Patient has been admitted with ID #"
317                  << temp_pat.get_id() << ui::endl << ui::endl;
318
319          message.setexit_button("Okay");
320          message.loop();
321          message.hide();
322
323          break;
324      }
325
326    case 2:
327      {
328          patient temp_patient;
329
330          while(1){
331              coord c(ui::scr_width / 3, ui::scr_height / 3);
332              box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);
333
334              long inp_pat_id;
335
336              login_box << ui::endl << "Patient Discharge"
337                      << ui::endl << "Enter patient ID : ";
```

82

```
338            login_box.setdefault(1);
339            login_box >> inp_pat_id;
340
341            login_box << ui::endl;
342            login_box.setexit_button("Submit");
343
344            login_box.loop();
345
346            login_box.hide();
347
348            temp_patient = hospital::get_patient_by_id(inp_pat_id);
349
350            if(temp_patient.get_id() == inp_pat_id){
351                break;
352                interface::clear_error();
353            }
354            else{
355                interface::error("Invalid Patient ID!!");
356                continue;
357            }
358        }
359
360        coord c(ui::scr_width / 3, ui::scr_height / 3);
361        box bill (c, ui::scr_width / 3, ui::scr_height / 2);
362
363        str tt;
364        sprintf(tt, "%d/%d/%d", temp_patient.get_admission_date(DAY),
365                                                temp_patient.
                                                    get_admission_date
                                                    (MONTH),
366                                                temp_patient.
                                                    get_admission_date
                                                    (YEAR));
367
368                                                interface::log_this(
                                                    tt);
369
370        int stay_len = abs( hospital::get_date_difference(
371                                            system::get_date(),
372                                            Date(
373                                                temp_patient.
                                                    get_admission_date
                                                    (DAY),
374                                                temp_patient.
                                                    get_admission_date
                                                    (MONTH),
375                                                temp_patient.
                                                    get_admission_date
                                                    (YEAR)
376                                            )
377                        ) );
378
379        bill << ui::endl << "Bill for " << temp_patient.get_name()
380            << ui::endl << "1. Stay for "
381            << stay_len << " days"  << ui::endl;
382
383        float total_bill;
384        bill.settcolor(GREEN);
385        bill << "$" << ( total_bill += hospital::calc_bill(stay_len) );
```

```
386
387                    for(int i = 0; i < 50; i++){
388                            transaction temp_trans = temp_patient.get_transaction(i);
389
390                            if( temp_trans.amount == 0 ){
391                                break;
392                            }
393
394                            bill << i+2 << ". " << temp_trans.reason << ui::endl;
395                            bill.settcolor(GREEN);
396                            bill << "\t$" << temp_trans.amount << ui::endl;
397                            bill.settcolor(ui::tcolor);
398
399                            total_bill += temp_trans.amount;
400                    }
401
402                bill.settcolor(CYAN);
403                bill << ui::endl << "Final bill : $" << total_bill;
404                bill.settcolor(ui::tcolor);
405                bill.setexit_button("Pay Bill");
406                bill.loop();
407                bill.hide();
408
409                hospital::discharge_patient(temp_patient);
410
411                break;
412            }
413
414        case 3:
415        {
416            int choice = 0;
417
418            patient temp_patient;
419
420            while(1){
421                coord c(ui::scr_width / 3, ui::scr_height / 3);
422                box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);
423                login_box.settcolor_input(YELLOW);
424
425                long inp_pat_id;
426
427                login_box << ui::endl << "Patient Data Alteration"
428                                << ui::endl << "Enter patient ID : ";
429                login_box.setdefault(1);
430                login_box >> inp_pat_id;
431
432                login_box << ui::endl;
433                login_box.setexit_button("Submit");
434
435                login_box.loop();
436
437                login_box.hide();
438
439                temp_patient = hospital::get_patient_by_id(inp_pat_id);
440
441                if(temp_patient.get_id() == inp_pat_id){
442                    break;
443                    interface::clear_error();
444                }
```

84

```
445                 else{
446                     interface::error("Invalid Patient ID!!");
447                     continue;
448                 }
449             }
450
451         while(choice < 1 || choice > 5){
452             coord c(ui::scr_width / 3, ui::scr_height / 3);
453             box menu (c, ui::scr_width / 3, ui::scr_height / 1.5);
454
455             menu << "Choose item to edit:"
456                     << ui::endl << "1. Disease/condition"
457                     << ui::endl << "2. Guardian name"
458                     << ui::endl << "3. Emergency contact"
459                     << ui::endl << "4. Emergency contact no."
460                     << ui::endl << "5. Insurance information"
461                     << ui::endl << ui::endl << "Choice : ";
462             menu.setdefault(1);
463             menu.settcolor_input(YELLOW);
464             menu >> choice;
465
466             menu << ui::endl;
467             menu.setexit_button("Submit");
468
469             menu.loop();
470             menu.hide();
471         }
472         switch(choice){
473             case 1:
474             {
475                 coord c(ui::scr_width / 3, ui::scr_height / 3);
476                 box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
477                 edit_screen.settcolor_input(YELLOW);
478
479                 edit_screen <<  "Enter disease/condition for " <<
                        temp_patient.get_name()
480                         << ui::endl << "Disease : ";
481                 disease temp = temp_patient.get_dis();
482                 edit_screen.setdefault(temp.name);
483                 edit_screen >> temp.name;
484                 edit_screen << ui::endl << "Type : ";
485                 edit_screen.setdefault(temp.type);
486                 edit_screen >> temp.type;
487                 edit_screen << ui::endl << "Type key : " << ui::endl
488                         << "0 - Brain\t1 - Heart" << ui::endl
489                         << "2 - Skin\t3 - Lung" << ui::endl
490                         << "4 - Bone\t5 - Eye" << ui::endl
491                         << "6 - Throat\t7 - Teeth" << ui::endl
492                         << "8 - Stomach\t9 - Blood" << ui::endl
493                         << "10 - General/full body condition"
494                         << ui::endl << ui::endl
495                         << "Symptoms" << ui::endl
496                         << "Symptom 1 : ";
497                 edit_screen.setdefault(temp.symptoms[0]);
498                 edit_screen >> temp.symptoms[0];
499                 edit_screen << ui::endl << "Symptom 2 : ";
500                 edit_screen.setdefault(temp.symptoms[1]);
501                 edit_screen >> temp.symptoms[1];
502                 edit_screen << ui::endl << "Symptom 3 : ";
```

```
503              edit_screen.setdefault(temp.symptoms[2]);
504              edit_screen >> temp.symptoms[2];
505              edit_screen << ui::endl << "Symptom 4 : ";
506              edit_screen.setdefault(temp.symptoms[3]);
507              edit_screen >> temp.symptoms[3];
508
509              edit_screen << ui::endl << ui::endl;
510              edit_screen.setexit_button("Submit");
511
512              edit_screen.loop();
513
514              edit_screen.hide();
515
516              temp_patient.set_dis(temp);
517              hospital::write_patient(temp_patient);
518
519              break;
520          }
521
522      case 2:
523          {
524              coord c(ui::scr_width / 3, ui::scr_height / 3);
525              box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
526              edit_screen.settcolor_input(YELLOW);
527
528              edit_screen <<  "Enter name of guardian for " << temp_patient
                     .get_name()
529                      << ui::endl << "Guardian Name : ";
530              str temp;
531              edit_screen.setdefault(temp_patient.get_guardian_name());
532              edit_screen >> temp;
533
534              edit_screen << ui::endl << ui::endl;
535              edit_screen.setexit_button("Submit");
536
537              edit_screen.loop();
538
539              edit_screen.hide();
540
541              temp_patient.set_guardian_name(temp);
542              hospital::write_patient(temp_patient);
543
544              break;
545          }
546
547      case 3:
548          {
549              coord c(ui::scr_width / 3, ui::scr_height / 3);
550              box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
551              edit_screen.settcolor_input(YELLOW);
552
553              edit_screen <<  "Enter emergency contact no. for " <<
                     temp_patient.get_name()
554                      << ui::endl << "Contact no. : ";
555              str temp;
556              edit_screen.setdefault(temp_patient.get_emergency_contact());
557              edit_screen >> temp;
558
559              edit_screen << ui::endl << ui::endl;
```

```
560                     edit_screen.setexit_button("Submit");
561
562                     edit_screen.loop();
563
564                     edit_screen.hide();
565
566                     temp_patient.set_emergency_contact(temp);
567                     hospital::write_patient(temp_patient);
568
569                     break;
570                 }
571
572             case 4:
573                 {
574                     coord c(ui::scr_width / 3, ui::scr_height / 3);
575                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
576                     edit_screen.settcolor_input(YELLOW);
577
578                     edit_screen <<  "Enter emergency contact no. for " <<
                            temp_patient.get_name()
579                                 << ui::endl << "Contact no. : ";
580                     phone temp;
581                     edit_screen.setdefault(temp_patient.get_emer_contact_no());
582                     edit_screen >> temp;
583
584                     edit_screen << ui::endl << ui::endl;
585                     edit_screen.setexit_button("Submit");
586
587                     edit_screen.loop();
588
589                     edit_screen.hide();
590
591                     temp_patient.set_emer_contact_no(temp);
592                     hospital::write_patient(temp_patient);
593
594                     break;
595                 }
596
597             case 5:
598                 {
599                     coord c(ui::scr_width / 3, ui::scr_height / 3);
600                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
601                     edit_screen.settcolor_input(YELLOW);
602
603                     edit_screen <<  "Enter insurance information for " <<
                            temp_patient.get_name()
604                                 << ui::endl << "Provider : ";
605                     insurance temp = temp_patient.get_insur_info();
606                     edit_screen.setdefault(temp.provider);
607                     edit_screen >> temp.provider;
608                     edit_screen << ui::endl << "Amount (in $) :";
609                     edit_screen.setdefault(temp.amount);
610                     edit_screen >> temp.amount;
611                     edit_screen << ui::endl << "Expiry date (DD/MM/YYYY):";
612                     char temp_date[11];
613                     edit_screen >> hospital::date_validity >> temp_date;
614
615                     edit_screen << ui::endl << ui::endl;
616                     edit_screen.setexit_button("Submit");
```

87

```
617
618                          edit_screen.loop();

619
620                          edit_screen.hide();

621
622                          temp.expiry = hospital::str_to_date(temp_date);
623                          temp_patient.set_insur_info(temp);
624                          hospital::write_patient(temp_patient);

625
626                          break;
627                      }

628
629                  }

630
631              break;
632          }
633          case 4:
634          {
635              break;
636          }
637      }
638  }
```

7. code/EMP.CPP

```cpp
1  #include "hosp.hpp"
2  #include "iface.hpp"
3  #include "emp.hpp"
4  #include "base.hpp"
5  #include <fstream.h>
6
7  ////////////////////////////////////
8  /// Function definitions for class employee
9
10 int employee::generate_id()
11 {
12     mkdir("employee");
13     unsigned long max_id;
14     ifstream fin;
15     fin.open("employee/max_id.dat", ios::binary);
16     if(!fin)
17     {
18         interface::log_this("employee::generate_id() : File max_id.dat not found
                 or error while loading file\nmax_id will be set to zero");
19         max_id = 0;
20     }
21     else
22     {
23         fin.read((char *) &max_id, sizeof(unsigned long));
24         if(fin.fail())
25         {
26             interface::log_this("employee::generate_id() : Error while reading
                     from file max_id.dat (fin.fail())\nFunction aborted");
27             id = 0;
28             return 0;
29         }
30     }
31     fin.close();
```

```
32      ++max_id;
33      id = max_id;
34      ofstream fout;
35      fout.open("employee/max_id.dat", ios::binary);
36      fout.write((char *) &max_id, sizeof(unsigned long));
37      if(fout.fail())
38      {
39          interface::log_this("employee::generate_id() : Error while writing to
                 file max_id.dat (fout.fail())\nFunction aborted");
40          return 0;
41      }
42      else
43          return 1;
44  }
45
46  int employee::generate_id_status = 1;
47
48  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)
49  {
50      if(!generate_id_status)
51      {
52          interface::error("ID cannot be generated for this employee. Check log");
53          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")
                 ;
54      }
55      else
56      {
57          employee::generate_id_status = generate_id();
58          id_to_emp i1(id, OTHERS);
59          if(!i1.status)
60          {
61              interface::error("ID not generated properly for this employee. Check
                     log");
62              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");
63          }
64          salary = inp6;
65          shift_start = inp7;
66          shift_end = inp8;
67      }
68  }
69
70  employee::employee() : person()
71  {
72      id = 0;
73  }
74
75  int employee::get_age()
76  {
77      ////////////////Updating age to present age//////////
78      set_dob(dob);        //This function is used here to invoke calc_age() in it
             only(because calc_age is directly not accessible)
79      void * temp = malloc( sizeof(doctor) );
80      if(temp != NULL && hospital::get_employee_by_id(id, temp))            //if
             employee's file exists on disk
```

89

```
81          {
82              hospital::write_employee( this );                    //overwrite that file
83          }
84          free(temp);
85          return age;
86  }
87
88  unsigned long employee::get_salary(){
89          return salary;
90  }
91
92  void employee::set_salary(unsigned long inp)
93  {
94          salary = inp;
95  }
96
97  Time employee::get_shift(int inp){
98          switch(inp){
99              case START:
100                 return shift_start;
101             case END:
102                 return shift_end;
103             default:
104                 return Time(0,0,0);
105         }
106 }
107
108 void employee::set_shift(int inp1, Time inp2)
109 {
110         switch (inp1)
111         {
112             case START:
113                 shift_start = inp2;
114                 return;
115             case END:
116                 shift_end = inp2;
117                 return;
118             default:
119                 return;
120         }
121 }
122
123 unsigned long employee::get_id()
124 {
125         return id;
126 }
127
128 transaction * employee::get_last_5_transactions()
129 {
130         transaction * t = (transaction *)malloc(5 * sizeof(transaction));
131         if(t == NULL)
132         {
133             interface::log_this("employee::get_last_5_transactions() :Not enough
                    memory to allocate buffer void * temp = malloc( sizeof(doctor) )\
                    nFunction aborted");
134             return NULL;
135         }
136         for(int i = 0; i < 5; ++i)
137         {
```

90

```
138            t[i] = transaction();
139        }
140        str temp;
141        switch( id_to_emp::convert(id) )
142        {
143            case INVALID:
144            {
145                char log_msg[300];
146                sprintf(log_msg, "employee::get_last_5_transactions() : The object
                        has invalid id (%lu)\nFunction aborted", id);
147                interface::log_this(log_msg);
148                free(t);
149                return NULL;
150            }
151            case DOCTOR:
152            {
153                sprintf(temp, "employee/doctor/%lu/trans.dat", id);
154                break;
155            }
156            case NURSE:
157            {
158                sprintf(temp, "employee/nurse/%lu/trans.dat", id);
159                break;
160            }
161            case RECEPTIONIST:
162            {
163                sprintf(temp, "employee/receptionist/%lu/trans.dat", id);
164                break;
165            }
166            case OTHERS:
167            {
168                sprintf(temp, "employee/%lu/trans.dat", id);
169                break;
170            }
171        }
172        ifstream fin ( temp ,ios::binary | ios::in | ios::nocreate | ios::ate);
173        if(!fin)
174        {
175            char log_msg[300];
176            sprintf(log_msg, "employee::get_last_5_transactions() : Failed to open
                    file trans.dat for id %lu\nFunction aborted", id);
177            interface::log_this(log_msg);
178            free(t);
179            return NULL;
180        }
181        int max_i, size_of_file = fin.tellg();
182        if( size_of_file >= ( 5 * sizeof(transaction) ) )
183        {
184            const int a = (-5) * sizeof(transaction);
185            fin.seekg(a, ios::end);
186            max_i = 5;
187        }
188        else
189        {
190            fin.seekg(0, ios::beg);
191            max_i = (int)( size_of_file / sizeof(transaction) );
192        }
193        for(i = 0; i < max_i && !fin.eof(); ++i)
194        {
```

```
195          fin.read((char *) (t+i), sizeof(transaction));
196          if(fin.fail())
197          {
198              char log_msg[300];
199              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);
200              interface::log_this(log_msg);
201              free(t);
202              return NULL;
203          }
204      }
205      fin.close();
206      return t;
207  }
208
209  int employee::get_generate_id_status()
210  {
211      return generate_id_status;
212  }
213
214  /////////////////////////////////////////////
215  //// Doctor, Nurse and Receptionist class member defs
216
217  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)
218  {
219      id_to_emp i1(get_id(), DOCTOR);
220      if(!i1.status)
221      {
222          interface::error("ID not generated properly for this employee. Check log"
                 );
223          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");
224      }
225      speciality[0] = inp10;
226      speciality[1] = inp11;
227
228      for(int i = 0; i < 10; i++){
229          patients[i] = 0;
230      }
231  }
232
233  doctor::doctor() : employee()
234  {
235      speciality[0] = speciality[1] = GEN + 1;     //storing an invalid value in
             speciality
236      for(int i = 0; i < 10; ++i)
237      {
238          patients[i] = 0;
239      }
240  }
241
242  int * doctor::get_speciality()
243  {
244      return speciality;
245  }
246
```

```
247  long * doctor::get_patients()
248  {
249      return patients;
250  }
251
252  void doctor::set_speciality(int inp[2])
253  {
254      speciality[0] = inp[0];
255      speciality[1] = inp[1];
256  }
257
258  void doctor::set_patients(long inp[10])
259  {
260      for(int i = 0; i < 10; ++i)
261      {
262          patients[i] = inp[i];
263      }
264  }
265
266  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)
267  {
268      id_to_emp i1(get_id(), NURSE);
269      if(!i1.status)
270      {
271          interface::error("ID not generated properly for this employee. Check log"
                 );
272          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");
273      }
274      for(int i = 0; i < 5; i++){
275          patients[i] = 0;
276      }
277  }
278
279  nurse::nurse() : employee()
280  {
281      for(int i = 0; i < 5; ++i)
282      {
283          patients[i] = 0;
284      }
285  }
286
287  long * nurse::get_patients()
288  {
289      return patients;
290  }
291
292  void nurse::set_patients(long inp[5])
293  {
294      for(int i = 0; i < 5; ++i)
295      {
296          patients[i] = inp[i];
297      }
298  }
299
300  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)
301  {
302      id_to_emp i1(get_id(), RECEPTIONIST);
303      if(!i1.status)
304      {
305          interface::error("ID not generated properly for this employee. Check log"
                  );
306          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");
307      }
308  }
309
310  receptionist::receptionist() : employee()
311  {}
312
313
314  /////////////////////////////////////
315  /// Function definitions for class id_to_emp
316
317  id_to_emp::id_to_emp(unsigned long inp1, int inp2)
318  {
319      status = 0;
320      id = inp1;
321      if(!id)
322      {
323          employee_type = INVALID;
324      }
325      else
326      {
327          employee_type = inp2;
328      }
329      mkdir("employee");
330      ofstream fout;
331      fout.open("employee/id_list.dat", ios::binary | ios::ate);
332      if(!fout)
333      {
334          interface::log_this("id_to_emp::id_to_emp() : File id_list.dat couldn't
                  be opened...\nFunction aborted");
335      }
336      else
337      {
338          fout.seekp(id * sizeof(id_to_emp), ios::beg);
339          fout.write((char *) this, sizeof(id_to_emp));
340          if(fout.fail())
341          {
342              interface::log_this("id_to_emp::id_to_emp() : Error while writing to
                      file id_list.dat (fout.fail())\nFunction aborted");
343          }
344          else
345          {
346              status = 1;
347          }
348      }
349  }
350
351  id_to_emp::id_to_emp()
352  {
353      id = employee_type = status = 0;
354  }
```

94

```
355
356  int id_to_emp::convert(unsigned long ID)
357  {
358       id_to_emp a;
359       ifstream fin;
360       fin.open("employee/id_list.dat", ios::binary);
361       if(!fin)
362       {
363           interface::log_this("id_to_emp::convert() : File id_list.dat not found!!"
                    );
364           return INVALID;
365       }
366       fin.seekg( (ID * sizeof(id_to_emp)) );
367       fin.read((char *) &a, sizeof(id_to_emp));
368       if(fin.fail())
369       {
370           interface::log_this("id_to_emp::convert() : Error while reading from file
                    id_list.dat (fin.fail())");
371           return INVALID;
372       }
373       fin.close();
374       if(a.id != ID)
375       {
376           interface::log_this("id_to_emp::convert() : (For dev only)Error in the
                    code... Recheck it!!");
377           return INVALID;
378       }
379       return a.employee_type;
380  }
```

8. code/PATIENT.CPP

```
1   #include "patient.hpp"
2   #include <fstream.h>
3
4   ///////FUNCTION DEFINITIONS FOR CLASS PATIENT/////////
5
6   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
7   {
8       fstream pat ("patient/max_id.dat", ios::in | ios::binary | ios::out);
9       long max_id;
10      pat.read( (char*) &max_id, sizeof(long) );
11      max_id++;
12
13      id = max_id;
14
15      pat.seekp(0);
16      pat.write( (char*) &max_id, sizeof(long) );
17      pat.close();
18
19      dis = inp6;
20      strcpy(guardian_name, inp7);
21      strcpy(emergency_contact, inp8);
22      strcpy(emer_contact_no, inp9);
23      insur_info = inp10;
```

```
24
25     admission_date = inp11;
26     Date dnow = system::get_date();
27
28     if( admission_date.day != dnow.day ||
29         admission_date.month != dnow.month ||
30         admission_date.year != dnow.year        )
31     {
32         set_dob(inp3, inp11);
33     }
34     for(int i = 0; i < 50; i++){
35         med[i][0] = med[i][1] = 0;
36     }
37
38     bill_amt = 0;    //bill_amt will be set by doctor after treatment
39     discharged = 0;
40 }
41
42 patient::patient()
43 {
44     id = 0;
45 }
46
47 long patient::get_id()
48 {
49     return id;
50 }
51
52 disease patient::get_dis()
53 {
54     return dis;
55 }
56
57 char* patient::get_guardian_name()
58 {
59     return guardian_name;
60 }
61
62 char* patient::get_emergency_contact()
63 {
64     return emergency_contact;
65 }
66
67 char* patient::get_emer_contact_no()
68 {
69     return emer_contact_no;
70 }
71
72 insurance patient::get_insur_info()
73 {
74     return insur_info;
75 }
76
77 int patient::get_admission_date(int inp)
78 {
79     switch(inp)
80     {
81         case DAY:
82             return admission_date.day;
```

```
 83            case MONTH:
 84                return admission_date.month;
 85            case YEAR:
 86                return admission_date.year;
 87            default:
 88                return 0;
 89        }
 90  }
 91
 92  int patient::get_discharge_date(int inp)
 93  {
 94        switch(inp)
 95        {
 96            case DAY:
 97                return discharge_date.day;
 98            case MONTH:
 99                return discharge_date.month;
100            case YEAR:
101                return discharge_date.year;
102            default:
103                return 0;
104        }
105  }
106
107  unsigned long patient::get_bill_amt()
108  {
109        return bill_amt;
110  }
111
112  int patient::get_med(int a, int b){
113        return med[a][b];
114  }
115
116  transaction patient::get_transaction(int trans_num){
117        str temp;
118        transaction trans;
119        sprintf(temp, "patient/%d/trans.dat", this->id);
120        ifstream patient_file ( temp , ios::out | ios::binary | ios::app );
121
122        int i = 0;
123        while ( i<=trans_num && patient_file ){
124            patient_file.read( (char*) &trans , sizeof(transaction) );
125            i++;
126        }
127        if( i!= trans_num ){
128            trans = transaction(0);
129        }
130        patient_file.close();
131        return trans;
132  }
133
134  void patient::set_dis(disease a)
135  {
136        dis = a;
137  }
138
139  void patient::set_guardian_name(char *a)
140  {
141        strcpy(guardian_name, a);
```

```
142  }
143
144  void patient::set_emergency_contact(char *a)
145  {
146      strcpy(emergency_contact, a);
147  }
148
149  void patient::set_emer_contact_no(char *a)
150  {
151      strcpy(emer_contact_no, a);
152  }
153
154  void patient::set_insur_info(insurance a)
155  {
156      insur_info = a;
157  }
158
159  void patient::set_admission_date(Date a)
160  {
161      admission_date = a;
162      set_dob(dob, admission_date);
163  }
164
165  void patient::set_bill_amt(unsigned long a)
166  {
167      bill_amt = a;
168  }
169
170  void patient::set_med(int a, int b, int c){
171      med[a][0] = b;
172      med[a][1] = c;
173  }
174
175  void patient::set_discharge_date(Date inp){
176      discharge_date = inp;
177  }
178
179  void patient::discharge(){
180      discharged = 1;
181  }
```

9. code/PROC.CPP

```
1   #include <iostream.h>
2   #include <fstream.h>
3
4   typedef char str[100];
5
6   struct procedure{
7       str name;
8       float cost;
9   };
10
11  void main(){
12      ofstream proc ("proc.dat" , ios::out | ios::binary | ios::app);
13      procedure a;
14      cin.ignore(1000, '\n');
15      cout << "\nName:";
```

```
16      cin.getline(a.name, 100, '\n');
17      cout << "\nCost:";
18      cin >> a.cost;
19      cout << endl << "Procedure : " << a.name << "  $" << a.cost << ".\nEnter next
            procedure:";
20      proc.write( (char*) &a , sizeof(a) );
21  }
```

10. code/UI/test.cpp

```
1   //No need to use ui::init() explicitly
2
3   #include "ui/ui.hpp"
4   #include "ui/test.hpp"
5
6   void test_weird_error()
7   {
8       int shit = 14;
9       box menu2(coord(2, 4), 40, 10 );
10      menu2 << "Enter your shit: ";
11      menu2 >> shit;
12      menu2.setexit_button("Submit my shit");
13      menu2.loop();
14
15      menu2.clear();
16      menu2 << "Your shit's coming up!" << ui::endl; getch();
17      menu2 << "Here's your shit: ";
18      menu2 << shit;
19      menu2 << ". Deal with it!" << ui::endl;
20
21      getch();
22  }
23
24  int exit_func()
25  {
26      char c = getch();
27      int x = wherex(), y = wherey();
28
29      gotoxy(1, ui::scr_height - 1);
30      if(c != '1')
31      {
32          cprintf("Returning 0"); getch();
33          gotoxy(x, y);
34          return 0;
35      }
36      else
37      {
38          cprintf("Returning 1"); getch();
39          gotoxy(x, y);
40          return 1;
41      }
42  }
43
44  void test_back()
45  {
46      box window;
47
48      int a, b;
```

99

```cpp
49        window << "Here's some sample text" << ui::endl;
50        window << "Enter some fake data I don't care about" << ui::endl;
51
52        window << "Fake #1: "; window >> a;
53        window << "Fake #2: "; window >> b;
54        window.setexit_button("A fake button");
55
56        window.setback_func(exit_func);
57
58        window.loop();
59    }
60
61    void test_all()
62    {
63        ui::clrscr();
64        box menu2(coord(2, 4), 40, 10 );
65
66        menu2.settcolor(GREEN);
67        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
68        menu2.settcolor(WHITE);
69        int menu2_height;
70        menu2_height = 10;
71    //  menu2.setheight(menu2_height);
72        menu2 << "View employee data" << ui::endl;
73        menu2.settcolor(ui::tcolor);
74    //  menu2 << "Enter employee's id: ";
75        unsigned long id;
76        menu2 >> id;
77        menu2 << ui::endl;
78        menu2.setexit_button("Submit");
79        menu2.loop();
80
81        menu2.clear();
82        menu2.setheight(15);
83        menu2.settcolor(GREEN);
84        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
85        menu2.settcolor(WHITE);
86        menu2 << "Employee Details: " << ui::endl;
87        menu2.settcolor(ui::tcolor);
88            getch();
89            menu2.hide();
90            getch();
91            menu2.display();
92            getch();
93        menu2 << "ID: " << 1 << ui::endl;
94            getch();
95            menu2.hide();
96            getch();
97            menu2.display();
98            getch();
99
100       char name[40], pwd[40];
101       int age;
102       long phn;
103       float amt;
104       char date[30];
105
106       box window;
107       window.settcolor(CYAN);
```

```
108        window << ui::centeralign << "LHOSPITAL";
109        window << ui::endl << ui::endl;
110        window.settcolor(ui::tcolor);
111        window.setfooter_tcolor(GREEN);
112
113        window << box::setheader << "28/10/2017"
114               << box::setheader << ui::rightalign << "11:45 PM"
115               << box::setfooter << ui::centeralign
116               << "Everything looks OK";
117
118        window << "Fill the following form: " << ui::endl;
119
120        coord c(ui::scr_width/4, ui::scr_height/3);
121        box b(c, ui::scr_width / 3, 10);
122
123        b.settcolor_input(YELLOW);
124        b << "Enter details: " << ui::endl
125          << "Name: "; b >> name;
126        b << "Age: "; b >> age;
127        b << "Phone num: "; b >> phn;
128        b << "Date: ";
129        b.setdefault("27/10/2017");
130        b >> date;
131        b << "Amount: "; b >> amt;
132        b << "Password: "; b >> box::setpassword >> pwd;
133
134        b.f.setvisibility_mode(frame::nosides);
135
136        b.f.display();
137        b.setexit_button("Submit");
138        b.loop();
139
140        b.hide();
141
142        window << "You entered the following data: " << ui::endl
143          << "Name: " << name << ui::endl
144          << "Age: " << age << ui::endl
145          << "Phone num: " << phn << ui::endl
146          << "Date: " << date << ui::endl
147          << "Amount: " << amt << ui::endl
148          << "Password: " << pwd << ui::endl;
149    }
150
151
152    void test_listlayout()
153    {
154        list_layout l;
155        l.setpos(coord(2,1));
156        l.setheight(6);
157
158        interactive *list[10];
159
160        //Setting the text boxes
161        for(int i = 0; i < 9; i++)
162        {
163            char s[] = {'A'+i, ':', ' ', '\0'};
164            l.settcolor(LIGHTGRAY);
165            l << coord(2, i + 1) << s;
166            l.settcolor(RED);
```

101

```cpp
167          list[i] = l.settext_box(coord(5, i + 1));
168      }
169
170      l.settcolor(LIGHTGRAY);
171      list[9] = l.setbutton(coord(3, i + 1), "Submit");
172
173      //Rudimentary scrolling
174      i = 100;
175      int j = 0;
176
177      int lines_scrolled = l.getlines_scrolled(),
178          height = l.getheight();
179
180      coord pos_topleft(2,1);
181      int y = pos_topleft.y;
182      while(i--)
183      {
184          coord c = list[j]->getpos();
185          if(c.y - lines_scrolled > height)
186          {
187              lines_scrolled = c.y - height;
188          }
189          else if(c.y - lines_scrolled < y)
190          {
191              lines_scrolled = c.y - y;
192          }
193
194          l.setlines_scrolled(lines_scrolled);
195          int response = list[j]->input(-lines_scrolled);
196
197          if(response == interactive::GOTONEXT)
198          {
199              if(j < 9) j++; else j = 0;
200          }
201          else if(response == interactive::GOTOPREV)
202          {
203              if(j > 0) j--; else j = 9;
204          }
205          else if(response == interactive::CLICKED)
206          {
207              coord init_pos(wherex(), wherey());
208              gotoxy(1, ui::scr_height-1);
209              cprintf("%s%d", "Clicked ", i);
210              gotoxy(init_pos.x, init_pos.y);
211          }
212      }
213  }
214
215  void test_textbox()
216  {
217      text_box t;
218      t.setpos(coord(1,1));
219      for(int i = 0; i < 5; i++)
220      {
221          int a = t.input();
222
223          int x = wherex(), y = wherey();
224          gotoxy(1, ui::scr_height-1);
225          if(a == interactive::GOTONEXT)
```

```cpp
226          {
227              cout << "GOTONEXT";
228          }
229          else if(a == interactive::GOTOPREV)
230          {
231              cout << "GOTOPREV";
232          }
233          else
234          {
235              cout << "UNDEFINED";
236          }
237
238          gotoxy(x, y);
239      }
240  }
241
242  void test_frame()
243  {
244      frame f;
245      f.display();
246
247      getch();
248
249      f << ui::top << 't'
250        << ui::left << 'l'
251        << ui::bottom << 'b'
252        << ui::right << 'r';
253
254      f.settcolor(LIGHTBLUE);
255
256      f.display();
257
258      getch();
259
260      f << (ui::top | ui::left) << (char) 201
261        << (ui::bottom | ui::left) << (char) 200
262        << (ui::top | ui::right) << (char) 187
263        << (ui::bottom | ui::right) << (char) 188
264        << ui::top << (char) 205
265        << ui::bottom << (char) 205
266        << ui::left << (char) 186
267        << ui::right << (char) 186;
268
269      f.settcolor(ui::tcolor);
270
271      f.display();
272
273      getch();
274
275      f.setheight(ui::scr_height/2);
276      getch();
277
278      f.setwidth(ui::scr_width/3);
279      getch();
280
281      f.setcorner_top_left(coord( (ui::scr_width-f.getwidth()) / 2, (ui::scr_height
            -f.getheight()) / 2));
282      getch();
283
```

```
284        f.setvisibility_mode(frame::nosides);
285   }
```

11. code/UI/interact.cpp

```cpp
1    #include "ui/ui.hpp"
2
3    string_node::string_node()
4    {
5        next = NULL;
6        prev = NULL;
7        data = '\0';
8    }
9
10   interactive::interactive()
11   {
12       prev = NULL;
13       next = NULL;
14   }
15
16   interactive::~interactive()
17   {
18       delete next;
19       next = NULL;
20       prev = NULL;
21   }
22
23   int interactive::input(int)
24   {
25       return -1;
26   }
27
28   void interactive::setoffset(int o)
29   {
30       offset = o;
31   }
32
33   int interactive::getoffset()
34   {
35       return offset;
36   }
37
38   int interactive::getkey()
39   {
40       char ch = getch();
41       switch(ch)
42       {
43           case 9:     return TAB;
44           case 13:    return ENTER;
45           case 8:
46           {
47               unsigned char far *key_state_byte
48                   = (unsigned char far*) 0x00400017;
49               int key_state = (int) *key_state_byte;
50
51               if(key_state & 2) return SHIFT_BACKSPACE;
52               else                  return BACKSPACE;
53           }
```

```
54          case 0:      break;
55          default:     return ch;
56      }
57
58      ch = getch();
59
60      unsigned char far *key_state_byte
61      = (unsigned char far*) 0x00400017;
62      int key_state = (int) *key_state_byte;
63
64      switch(ch)
65      {
66          case 72:     return UP;
67          case 80:     return DOWN;
68          case 75:     return LEFT;
69          case 77:     return RIGHT;
70          case 15:     if (key_state & 2) return SHIFT_TAB;
71          //            ^^ Checks if shift was pressed
72          case 83:     return DELETE;
73          case 71:     return HOME;
74          case 79:     return END;
75      }
76
77      return -1;
78 }
```

12. code/UI/uibase.cpp

```
1  #include "ui/ui.hpp"
2  #include "iface.hpp"
3
4  int init_lib_ui::counter = 0;
5
6  init_lib_ui::init_lib_ui()
7  {
8      if(counter++ == 0)
9      {
10          ui::init();
11      }
12 }
13
14 int manipulator::index = 0;
15
16 manipulator::manipulator()
17 {
18      own_index = index;
19      index++;
20 }
21
22 int manipulator::operator==(manipulator m)
23 {
24      return own_index == m.own_index;
25 }
26
27 int ui::scr_height = 0,
28     ui::scr_width = 0,
29     ui::tcolor = LIGHTGRAY,
30     ui::bcolor = BLACK;
```

105

```cpp
31  manipulator ui::endl,
32              ui::centeralign,
33              ui::rightalign;
34
35  void ui::init()
36  {
37      set_new_handler(ui::my_new_handler);
38
39      ui::clrscr();
40
41      textcolor(ui::tcolor);
42      textbackground(ui::bcolor);
43
44      struct text_info info;
45      gettextinfo(&info);
46
47      //height and width of screen
48      scr_width = (int) info.screenwidth;
49      scr_height = (int) info.screenheight;
50  }
51
52  void ui::clrscr()
53  {
54      ::clrscr();
55  }
56
57  void ui::my_new_handler()
58  {
59      interface::log_this("Error in allocating memory. Exiting...");
60      exit(1);
61  }
62
63  coord::coord(int X, int Y)
64  {
65      x = X;
66      y = Y;
67  }
68
69  coord & coord::operator+=(coord b)
70  {
71      x += b.x;
72      y += b.y;
73
74      return *this;
75  }
76
77  coord & coord::operator-=(coord b)
78  {
79      x -= b.x;
80      y -= b.y;
81
82      return *this;
83  }
84
85  coord coord::operator+(coord b)
86  {
87      coord temp = *this;
88      return temp += b ;
89  }
```

```
90
91  coord coord::operator−(coord b)
92  {
93      coord temp = *this;
94      return temp −= b;
95  }
```

13. code/UI/frame.cpp

```
1   #include "ui/ui.hpp"
2
3   int frame::convert(int param)
4   {
5       if(param & ui::top)
6       {
7           if(param & ui::left)
8           {
9               return 0;
10          }
11          else if(param & ui::right)
12          {
13              return 1;
14          }
15          else
16          {
17              return 2;
18          }
19      }
20      else if(param & ui::bottom)
21      {
22          if(param & ui::left)
23          {
24              return 3;
25          }
26          else if(param & ui::right)
27          {
28              return 4;
29          }
30          else
31          {
32              return 5;
33          }
34      }
35      else if(param & ui::left)
36      {
37          return 6;
38      }
39      else if(param & ui::right)
40      {
41          return 7;
42      }
43
44      return −1;
45  }
46
47  void frame::setside_visibility(int side, int visib)
48  {
49      if( visib != 0 && visib != 1)
```

```
50                return;      //No effect for invalid visibility
51
52        if(side & ui::all)
53        {
54            for(int i = 0; i < 8; i++)
55                sides_visibility[i] = visib;
56            return;
57        }
58
59        int a = frame::convert(side);
60        if(a == −1) return; //−1 indicates invalid side
61
62        sides_visibility[a] = visib;
63    }
64
65    int frame::getside_visibility(int side)
66    {
67        int a = convert(side);
68
69        if(a == −1) return −1; //Wrong side selected
70
71        return sides_visibility[a];
72    }
73
74
75    frame::frame(coord topleft, int w, int h)
76    {
77        for(int i = 0; i < 8; i++)
78        {
79            border_chars[i] = '*';
80            sides_visibility[i] = 1;
81        }
82        tcolor = ui::tcolor;
83        bcolor = ui::bcolor;
84        frame_visibility = 0;
85        height = h;
86        width = w;
87        state = 0;
88        corner_top_left = topleft;
89    }
90
91    void frame::display()
92    {
93        print(1);
94    }
95
96    void frame::hide()
97    {
98        print(0);
99    }
100
101   void frame::print(int param)
102   {
103       textcolor(frame::tcolor);
104       textbackground(frame::bcolor);
105
106       char visible_chars[8];
107       frame_visibility = param;
108
```

```
109        int x = corner_top_left.x,
110            y = corner_top_left.y;
111
112        int arr[] = {
113                    ui::top,
114                    ui::bottom,
115                    ui::left,
116                    ui::right,
117                    ui::top | ui::left,
118                    ui::top | ui::right,
119                    ui::bottom | ui::left,
120                    ui::bottom | ui::right
121                };
122
123        char &top = visible_chars[0],
124             &bottom = visible_chars[1],
125             &left = visible_chars[2],
126             &right = visible_chars[3],
127             &top_left = visible_chars[4],
128             &top_right = visible_chars[5],
129             &bottom_left = visible_chars[6],
130             &bottom_right = visible_chars[7];
131
132        for(int i = 0; i < 8; i++)
133        {
134            if(param == 1 && getside_visibility(arr[i]))
135            {
136                visible_chars[i] = getborder_char(arr[i]);
137            }
138            else
139            {
140                visible_chars[i] = ' ';
141            }
142        }
143
144        gotoxy(x, y);
145
146        cprintf("%c", top_left);
147
148        for(i = 1; i < width - 1; i++)
149        {
150            cprintf("%c", top);
151        }
152        cprintf("%c", top_right);
153
154        for(i = 1; i < height - 1; i++)
155        {
156            gotoxy(x, y + i); cprintf("%c", left);
157            gotoxy(x + width - 1, y + i); cprintf("%c", right);
158        }
159
160        gotoxy(x, y + height - 1);
161        cprintf("%c", bottom_left);
162        for(i = 1; i < width - 1; i++)
163        {
164            cprintf("%c", bottom);
165        }
166        cprintf("%c", bottom_right);
167
```

```
168        gotoxy(corner_top_left.x, corner_top_left.y);
169
170        textcolor(ui::tcolor);
171   }
172
173   void frame::setvisibility_mode(int param)
174   {
175        frame::setside_visibility(frame::all, 1);
176        if(param & nosides)
177        {
178            frame::setside_visibility(ui::left, 0);
179            frame::setside_visibility(ui::right, 0);
180        }
181        frame::display();
182   }
183
184   //Operator << is used to set border char
185   frame & frame::operator<<(int side)
186   {
187        int a = frame::convert(side);
188
189        if(a == -1) return *this; //-1 indicates error
190
191        state = a;
192
193        return *this;
194   }
195
196   frame & frame::operator<<(char border_char)
197   {
198        border_chars[frame::state] = border_char;
199        return *this;
200   }
201
202   int frame::getheight()
203   {
204        return height;
205   }
206
207   int frame::getwidth()
208   {
209        return width;
210   }
211
212   coord frame::getcorner_top_left()
213   {
214        return frame::corner_top_left;
215   }
216
217   int frame::getframe_visibility()
218   {
219        return frame_visibility;
220   }
221
222   int frame::gettcolor()
223   {
224        return tcolor;
225   }
226
```

```
227  int frame::getbcolor()
228  {
229      return bcolor;
230  }
231
232  char frame::getborder_char(int side)
233  {
234      int a = convert(side);
235
236      if(a == −1) return '\0'; //Error
237
238      return frame::border_chars[a];
239  }
240
241  void frame::setheight(int h)
242  {
243      if(h > ui::scr_height) return;
244
245      hide();
246      frame::height = h;
247      display();
248  }
249
250  void frame::setwidth(int w)
251  {
252      if(w > ui::scr_width) return;
253
254      hide();
255      frame::width = w;
256      display();
257  }
258
259  void frame::settcolor(int c)
260  {
261      tcolor = c;
262      display();
263  }
264
265  void frame::setbcolor(int b)
266  {
267      bcolor = b;
268      display();
269  }
270
271  void frame::setcorner_top_left(coord c)
272  {
273      hide();
274      frame::corner_top_left = c;
275      display();
276  }
```

14. code/UI/box.cpp

```
1  #include "ui/ui.hpp"
2  #include "iface.hpp"
3
4  line::line()
5  {
```

```
6        strcpy(left, "");
7        strcpy(middle, "");
8        strcpy(right, "");
9        width = ui::scr_width - 2;
10       tcolor = ui::tcolor;
11       bcolor = ui::bcolor;
12       corner_top_left = coord(0,0);
13   }
14
15   void line::display()
16   {
17       print(1);
18   }
19
20   void line::hide()
21   {
22       print(0);
23   }
24
25   void line::clear()
26   {
27       hide();
28       strcpy(left, "");
29       strcpy(middle, "");
30       strcpy(right, "");
31       display();
32   }
33
34   void line::print(int mode)
35   {
36       coord curr_pos = coord(wherex(), wherey()),
37       &ctl = corner_top_left;
38       gotoxy(ctl.x, ctl.y);
39       textcolor(tcolor);
40       textbackground(bcolor);
41
42       if(mode == 1)
43       {
44           cprintf("%s", left);
45       }
46       else
47       {
48           for(int i = 0; i < strlen(left); i++)
49           {
50               cprintf(" ");
51           }
52       }
53
54       gotoxy(ctl.x + (width - strlen(middle)) / 2,
55                   wherey());
56       if(mode == 1)
57       {
58           cprintf("%s", middle);
59       }
60       else
61       {
62           for(int i = 0; i < strlen(middle); i++)
63           {
64               cprintf(" ");
```

112

```
65                }
66            }
67
68        gotoxy(ctl.x + width − strlen(right), wherey());
69        if(mode == 1)
70        {
71            cprintf("%s", right);
72        }
73        else
74        {
75            for(int i = 0; i < strlen(right); i++)
76            {
77                cprintf(" ");
78            }
79        }
80
81        gotoxy(curr_pos.x, curr_pos.y);
82    }
83
84    int default_back_func()
85    {
86        return 0;
87    }
88
89    int box::wrap(char str[], int length, int return_one_line)
90    {
91        int num_lines = 1;
92        char out_str[300] = "";
93
94        int pos_old_newline = −1,
95            pos_curr_newline = −1;
96
97        int len_str = strlen(str);
98
99        //Iterating upto len_str because the '\0' at the end of the string
100       //would be interpreted as a newline
101       for(int i = 0; i <= len_str; i++)
102       {
103           if(str[i] == '\n' || i == len_str)
104           {
105               pos_old_newline = pos_curr_newline;
106               pos_curr_newline = i;
107
108               if(pos_curr_newline != len_str) num_lines++;
109
110               int chars_read = 0,
111                   read,
112                   written = 0;
113
114               char word[30];
115
116               str[pos_curr_newline] = '\0';
117
118               char *line = str + pos_old_newline + 1;
119               while(sscanf(line + chars_read, "%s%n", word, &read) > 0)
120               {
121                   int word_len = strlen(word);
122                   if(written + word_len  > length)
123                   {
```

113

```
124                            num_lines++;
125                            sprintf(out_str + strlen(out_str), "\n%s ", word);
126                            written = word_len + 1;
127                        }
128                    else if(written + word_len < length)
129                        {
130                            sprintf(out_str + strlen(out_str), "%s ", word);
131                            written += word_len + 1;
132                        }
133                    else //Not to add the space at the end if the line just completes
134                        {
135                            sprintf(out_str + strlen(out_str), "%s", word);
136                            written += word_len;
137                        }
138
139                        chars_read += read;
140                    }
141
142                if(pos_curr_newline != len_str)
143                        sprintf(out_str + strlen(out_str), "\n");
144                        str[pos_curr_newline] = '\n';
145            }
146        }
147
148    //An extra space is at the end of the string which has to be removed
149    //out_str[strlen(out_str) - 1] = '\0';
150    sprintf(str, "%s", out_str);
151
152    if(!return_one_line)    return num_lines;
153
154    len_str = strlen(str);
155
156    for(i = 0; i <= len_str; i++)
157    {
158        if(i == len_str)
159        {
160            break;
161        }
162        else if(str[i] == '\n')
163        {
164            str[i] = '\0';
165            break;
166        }
167    }
168
169    return num_lines;
170 }
171
172 void box::set_tbox(int data_type, void *ptr)
173 {
174    text_box *new_tbox;
175
176    if(data_type == info_tbox::PASSWORD)
177    {
178        new_tbox =
179            (text_box *) layout.settext_box(pos_pointer, 1);
180    }
181    else
182    {
```

```
183         new_tbox =
184             (text_box *) layout.settext_box(pos_pointer);
185     }
186
187     if(default_toggle)
188     {
189         default_toggle = 0;
190         new_tbox -> setstr(default_text);
191     }
192
193     pos_pointer.y++;
194     pos_pointer.x = layout.getcorner_top_left().x;
195
196     list_interactive[index_interactive]
197         = (interactive *) new_tbox;
198     info_tbox &t = list_tbox[index_tbox];
199     index_interactive++;
200     index_tbox++;
201
202     t.tbox = new_tbox;
203     t.type = data_type;
204     t.data_store = ptr;
205     t.validator = validation::getvalidator(data_type, temp_validator);
206
207     temp_validator = NULL;
208 }
209
210 manipulator box::setheader,
211             box::setfooter,
212             box::setpassword;
213
214 box::box(coord c, int w, int h) : f(c, w, h)
215 {
216     width = w;
217     height = h;
218     padding = 1;
219
220     corner_top_left = c;
221
222     f << (ui::top | ui::left) << (char) 201
223       << (ui::bottom | ui::left) << (char) 200
224       << (ui::top | ui::right) << (char) 187
225       << (ui::bottom | ui::right) << (char) 188
226       << ui::top << (char) 205
227       << ui::bottom << (char) 205
228       << ui::left << (char) 186
229       << ui::right << (char) 186;
230
231     layout.setwidth(w - 2 - 2 * padding);
232     layout.setheight(h - 2 - 2 * padding);
233     //                    ^bcoz of frame
234     layout.setcorner_top_left(c +
235         coord(1 + padding, 1 + padding));
236
237     pos_pointer = layout.getcorner_top_left();
238
239     for(int i = 0; i < 30; i++)
240     {
241         list_interactive[i] = NULL;
```

```
242        }
243        exit_btn = NULL;
244        index_interactive = index_tbox = 0;
245        center_toggle = 0;
246        default_toggle = 0;
247        right_toggle = 0;
248        header_toggle = 0;
249        footer_toggle = 0;
250        password_toggle = 0;
251        strcpy(default_text, "");
252        temp_validator = NULL;
253
254        header.width = footer.width = w - 2;
255        header.corner_top_left = c + coord(1,0);
256        footer.corner_top_left = c + coord(0, h-1);
257
258        back_func = default_back_func;
259
260        f.display();
261   }
262
263   coord box::getcorner_top_left()
264   {
265        return corner_top_left;
266   }
267
268   int box::getheight()
269   {
270        return height;
271   }
272
273   int box::getwidth()
274   {
275        return width;
276   }
277
278   int box::getpadding()
279   {
280        return padding;
281   }
282
283   void box::setcorner_top_left(coord c)
284   {
285        corner_top_left = c;
286        f.setcorner_top_left(c);
287        c += coord(1 + padding, 1 + padding);
288        layout.setcorner_top_left(c);
289
290        pos_pointer = c;
291   }
292
293   void box::setheight(int h)
294   {
295        height = h;
296        f.setheight(h);
297        layout.setheight(h - 2 - 2 * padding);
298   }
299
300   void box::setpadding(int p)
```

116

```
301  {
302      hide();
303      padding = p;
304      setheight(height);
305      display();
306  }
307
308  void box::settcolor(int c)
309  {
310      layout.settcolor(c);
311  }
312
313  void box::setbcolor(int c)
314  {
315      layout.setbcolor(c);
316  }
317
318  void box::settcolor_selected(int c)
319  {
320      layout.settcolor_selected(c);
321  }
322
323  void box::setbcolor_selected(int c)
324  {
325      layout.setbcolor_selected(c);
326  }
327
328  void box::settcolor_input(int c)
329  {
330      layout.settcolor_input(c);
331  }
332
333  void box::setbcolor_input(int c)
334  {
335      layout.setbcolor_input(c);
336  }
337
338  void box::setback_func( int(*f)(void) )
339  {
340      back_func = f;
341  }
342
343  box & box::operator<< (char *inp_str)
344  {
345      char string[100];
346      char *str = string;
347      strcpy(string, inp_str);
348
349      coord c = layout.getcorner_top_left();
350
351      if(header_toggle || footer_toggle)
352      {
353          line *lp;
354          if(header_toggle)
355          {
356              header_toggle = 0;
357              lp = &header;
358          }
359          if(footer_toggle)
```

117

```
360                 {
361                     footer_toggle = 0;
362                     lp = &footer;
363                 }
364             line &l = *lp;
365
366             int len = strlen(string);
367             if(center_toggle)
368             {
369                 center_toggle = 0;
370                 if(len <= l.width)
371                 {
372                     if((l.width - len) / 2 > strlen(l.left))
373                     {
374                         strcpy(l.middle, string);
375                     }
376                 }
377             }
378             else if(right_toggle)
379             {
380                 right_toggle = 0;
381                 if(len <= l.width)
382                 {
383                     if(len < (l.width - strlen(l.middle)) / 2)
384                     {
385                         strcpy(l.right, string);
386                     }
387                 }
388             }
389             else
390             {
391                 if(len < (l.width - strlen(l.middle)) / 2)
392                 {
393                     strcpy(l.left, string);
394                 }
395             }
396
397             //Printing the newly set line
398             l.hide();
399             l.display();
400
401             return *this;
402         }
403
404         if(center_toggle)
405         {
406             int len = strlen(string);
407             center_toggle = 0;
408             if(len <= layout.getwidth())
409             {
410                 int x_center_pos =
411                     c.x + (layout.getwidth() - len) / 2;
412
413                 if(pos_pointer.x > x_center_pos)
414                 {
415                     pos_pointer.y++;
416                 }
417                 pos_pointer.x = x_center_pos;
418                 layout << pos_pointer << str;
```

```
419              pos_pointer.x += len;
420              return *this;
421          }
422      }
423      else if(right_toggle)
424      {
425          int len = strlen(string);
426          right_toggle = 0;
427          if(len <= layout.getwidth())
428          {
429              int x_right_pos =
430                  c.x + (layout.getwidth() - len);
431
432              if(pos_pointer.x > x_right_pos)
433              {
434                  pos_pointer.y++;
435              }
436              pos_pointer.x = x_right_pos;
437              layout << pos_pointer << str;
438              pos_pointer.y++;
439              pos_pointer.x = c.x;
440              return *this;
441          }
442      }
443
444      int num_lines;
445
446      if(pos_pointer.x != c.x)
447      {
448          int remaining_space = layout.getwidth() -
449          (pos_pointer.x - layout.getcorner_top_left().x);
450          char s[100];
451          strcpy(s, str);
452          num_lines = wrap(s, remaining_space, 1);
453
454          layout << pos_pointer << s;
455
456          if(num_lines > 1)
457          {
458              pos_pointer.x = c.x;
459              pos_pointer.y++;
460          }
461          else
462          {
463              pos_pointer.x += strlen(s);
464          }
465
466          if (num_lines == 1 ||
467              str[strlen(str) - 1] == '\n')   return *this;
468
469          str += strlen(s); //There's an extra space at the end of s
470      }
471
472      num_lines = wrap(str, layout.getwidth());
473
474      int len_str = strlen(str),
475          pos_curr_newline = -1,
476          chars_to_forward = 0;
477
```

```cpp
478        for(int i = 0; i < len_str; i++)
479        {
480            if(str[i] == '\n')
481            {
482                pos_curr_newline = i;
483
484                str[pos_curr_newline] = '\0';
485                layout << pos_pointer << str + chars_to_forward;
486                pos_pointer.y++;
487
488                chars_to_forward +=
489                    strlen(str + chars_to_forward) + 1;
490            }
491        }
492
493        if(i == len_str − 1)    return *this;
494
495        layout << pos_pointer << str + chars_to_forward;
496        pos_pointer.x += strlen(str + chars_to_forward);
497
498        return *this;
499    }
500
501    box & box::operator<<(char ch)
502    {
503        char str[] = {ch, '\0'};
504        return (*this) << str;
505    }
506
507    box & box::operator<<(int i)
508    {
509        return (*this) << (long) i;
510    }
511
512    box & box::operator<<(long l)
513    {
514        char str[100];
515        sprintf(str,"%ld", l);
516        return (*this) << str;
517    }
518
519    box & box::operator<<(unsigned long ul)
520    {
521        char str[100];
522        sprintf(str, "%lu", ul);
523        return (*this) << str;
524    }
525
526    box & box::operator<<(double d)
527    {
528        char str[100];
529        sprintf(str, "%g", d);
530        return (*this) << str;
531    }
532
533    box & box::operator<<(float f)
534    {
535        char str[100];
536        sprintf(str, "%f", f);
```

120

```cpp
537         return (*this) << str;
538     }
539
540     box & box::operator<<(manipulator m)
541     {
542         if(m == ui::endl)
543         {
544             pos_pointer.y++;
545             pos_pointer.x = layout.getcorner_top_left().x;
546         }
547         else if(m == ui::centeralign)
548         {
549             center_toggle = 1;
550         }
551         else if(m == ui::rightalign)
552         {
553             right_toggle = 1;
554         }
555         else if(m == box::setheader)
556         {
557             header_toggle = 1;
558         }
559         else if(m == box::setfooter)
560         {
561             footer_toggle = 1;
562         }
563         return *this;
564     }
565
566     box & box::operator>>(char *&s)
567     {
568         if(password_toggle)
569         {
570             password_toggle = 0;
571             set_tbox(info_tbox::PASSWORD, (void *) s);
572         }
573         else
574         {
575             set_tbox(info_tbox::STRING, (void *) s);
576         }
577         return *this;
578     }
579
580     box & box::operator>>(char &ch)
581     {
582         set_tbox(info_tbox::CHAR, (void *) &ch);
583         return *this;
584     }
585
586     box & box::operator>>(int &i)
587     {
588         set_tbox(info_tbox::INT, (void *) &i);
589         return *this;
590     }
591
592     box & box::operator>>(long &l)
593     {
594         set_tbox(info_tbox::LONG, (void *) &l);
595         return *this;
```

```
596  }
597
598  box & box::operator>>(unsigned long &ul)
599  {
600      set_tbox(info_tbox::UNSIGNED_LONG, (void *) &ul);
601      return *this;
602  }
603
604  box & box::operator>>(double &d)
605  {
606      set_tbox(info_tbox::DOUBLE, (void *) &d);
607      return *this;
608  }
609
610  box & box::operator>>(float &f)
611  {
612      set_tbox(info_tbox::FLOAT, (void *) &f);
613      return *this;
614  }
615
616  box & box::operator>>(manipulator m)
617  {
618      if(m == box::setpassword)
619      {
620          password_toggle = 1;
621      }
622      return *this;
623  }
624
625  box & box::operator>>(int (*f)(const char *))
626  {
627      temp_validator = f;
628      return *this;
629  }
630
631  void box::setexit_button(char *str)
632  {
633      coord c = layout.getcorner_top_left();
634      if(pos_pointer.x != c.x)
635          pos_pointer.y++;
636
637      pos_pointer.x = c.x + (layout.getwidth() - strlen(str)) / 2;
638
639      button * new_btn =
640          (button *) layout.setbutton(pos_pointer, str);
641
642      pos_pointer.y++;
643      pos_pointer.x = c.x;
644
645      exit_btn = new_btn;
646      list_interactive[index_interactive]
647          = (interactive *) new_btn;
648      index_interactive++;
649  }
650
651  void box::setdefault(char *s)
652  {
653      default_toggle = 1;
654      strcpy(default_text, s);
```

122

```
655    }
656
657    void box::setdefault(char c)
658    {
659        char s[] = {c, '\0'};
660        setdefault(s);
661    }
662
663    void box::setdefault(int i)
664    {
665        setdefault( (long) i);
666    }
667
668    void box::setdefault(long l)
669    {
670        char s[100];
671        sprintf(s, "%ld", l);
672        setdefault(s);
673    }
674
675    void box::setdefault(unsigned long ul)
676    {
677        char s[100];
678        sprintf(s, "%lu", ul);
679        setdefault(s);
680    }
681
682    void box::setdefault(double d)
683    {
684        char s[100];
685        sprintf(s, "%g", d);
686        setdefault(s);
687    }
688
689    void box::setdefault(float f)
690    {
691        char s[100];
692        sprintf(s, "%f", f);
693        setdefault(s);
694    }
695
696    void box::loop()
697    {
698        int j = 0,
699        lines_scrolled = layout.getlines_scrolled(),
700        height = layout.getheight(),
701        index_last_interactive = index_interactive - 1,
702        &ili = index_last_interactive;
703        int temp_tbox_color, temp_index = -1;
704
705        inf_loop:
706        while(1)
707        {
708            coord c = list_interactive[j]->getpos(),
709                  ctl = layout.getcorner_top_left();
710            if(c.y - ctl.y - lines_scrolled + 1 > height)
711            {
712                lines_scrolled = c.y - ctl.y - height + 1;
713            }
```

123

```
714        else if(c.y − lines_scrolled < ctl.y)
715        {
716            lines_scrolled =
717                c.y − ctl.y;
718        }
719
720        layout.setlines_scrolled(lines_scrolled);
721        int response =
722            list_interactive[j]−>input(−lines_scrolled);
723
724        if(response == interactive::GOTONEXT)
725        {
726            if(j < ili) j++; else j = 0;
727        }
728        else if(response == interactive::GOTOPREV)
729        {
730            if(j > 0) j−−; else j = ili;
731        }
732        else if(response == interactive::CLICKED)
733        {
734            break;
735        }
736        else if(response == interactive::BACK && back_func())
737        {
738            return;
739        }
740    }
741
742    interface::clear_error();
743    if(temp_index != −1)
744    {
745        list_tbox[temp_index].tbox−>settcolor(temp_tbox_color);
746    }
747    for(int i = 0; i < index_tbox; i++)
748    {
749        if(list_tbox[i].setdata() == 0)
750        {
751            interface::error("INVALID INPUT!");
752            temp_tbox_color = list_tbox[i].tbox−>gettcolor();
753            list_tbox[i].tbox−>settcolor(RED);
754            temp_index = i;
755            goto inf_loop;
756        }
757    }
758 }
759
760 void box::display()
761 {
762    layout.display();
763    f.display();
764    header.display();
765    footer.display();
766 }
767
768 void box::hide()
769 {
770    layout.hide();
771    f.hide();
772    header.hide();
```

```
773        footer.hide();
774    }
775
776    void box::clear()
777    {
778        layout.hide();
779        layout.clear();
780        pos_pointer = layout.getcorner_top_left();
781        index_interactive = index_tbox = 0;
782        exit_btn = NULL;
783        f.display();
784    }
785
786    void box::setheader_tcolor(int c)
787    {
788        header.tcolor = c;
789    }
790
791    void box::setfooter_tcolor(int c)
792    {
793        footer.tcolor = c;
794    }
795
796    void box::clear_header()
797    {
798        header.clear();
799        f.display();
800        footer.display();
801    }
802
803    void box::clear_footer()
804    {
805        footer.clear();
806        f.display();
807        header.display();
808    }
```

15. code/UI/validation.cpp

```
1    #include "ui/ui.hpp"
2
3    int validation::vint(const char *str)
4    {
5        if(!validation::vlong(str)) return 0;
6
7        char *end;
8        long l = strtol(str, &end, 10);
9        if(l > INT_MAX || l < INT_MIN)
10       {
11           return 0;
12       }
13
14       return 1;
15   }
16
17   int validation::vlong(const char *str)
18   {
19       char *end;
```

125

```
20      long val = strtol(str, &end, 10);
21
22      if (errno == ERANGE || (errno != 0 && val == 0))
23      {
24          //If the converted value would fall
25          //out of the range of the result type.
26          return 0;
27      }
28      if (end == str)
29      {
30          //No digits were found.
31          return 0;
32      }
33
34      //Check if the string was fully processed.
35      return *end == '\0';
36  }
37
38  int validation::vunsigned_long(const char *str)
39  {
40      char *end;
41      unsigned long val = strtoul(str, &end, 10);
42
43      if (errno == ERANGE || (errno != 0 && val == 0))
44      {
45          return 0;
46      }
47      if (end == str || *end != '\0')
48      {
49          return 0;
50      }
51
52      int len = strlen(str);
53      for(int i = 0; i < len && isspace(str[i]); i++);
54
55      if(str[i] == '−')  return 0;
56
57      return 1;
58  }
59
60  int validation::vstring(const char *str)
61  {
62      return 1;
63  }
64
65  int validation::vchar(const char *str)
66  {
67      if(strlen(str) == 1 && isalnum(str[0]))
68      {
69          return 1;
70      }
71      return 0;
72  }
73
74  int validation::vdouble(const char *str)
75  {
76      char *end;
77      double val = strtod(str, &end);
78
```

126

```
79      if (errno == ERANGE)
80      {
81          //If the converted value would fall
82          //out of the range of the result type.
83          return 0;
84      }
85      if (end == str)
86      {
87          //No digits were found.
88          return 0;
89      }
90
91      return *end == '\0';
92  }
93
94  int validation::vfloat(const char *str)
95  {
96      return validation::vdouble(str);
97  }
98
99  validator_f validation::getvalidator
100                 (int type, validator_f v)
101  {
102      if(v != NULL) return v;
103
104      switch(type)
105      {
106          case info_tbox::INT:
107              return validation::vint;
108          case info_tbox::LONG:
109              return validation::vlong;
110          case info_tbox::UNSIGNED_LONG:
111              return validation::vunsigned_long;
112          case info_tbox::STRING:
113          case info_tbox::PASSWORD:
114              return validation::vstring;
115          case info_tbox::CHAR:
116              return validation::vchar;
117          case info_tbox::DOUBLE:
118              return validation::vdouble;
119          case info_tbox::FLOAT:
120              return validation::vfloat;
121      }
122
123      //TODO: log undefined behaviour
124      return NULL;
125  }
```

16. code/UI/llayout.cpp

```
1   #include "ui/ui.hpp"
2
3   list_layout_node::list_layout_node()
4   {
5       next = NULL;
6       tcolor = ui::tcolor;
7       bcolor = ui::bcolor;
8       strcpy(str, "");
```

```cpp
 9        print_type = DEFAULT;
10   }
11
12   list_layout_node::~list_layout_node()
13   {
14        delete next;
15        next = NULL;
16   }
17
18   //Setters
19   void list_layout_node::setnext(list_layout_node *n)
20   {
21        next = n;
22   }
23
24   void list_layout_node::setpos(coord p)
25   {
26        pos = p;
27   }
28
29   void list_layout_node::settcolor(int t)
30   {
31        tcolor = t;
32   }
33
34   void list_layout_node::setbcolor(int b)
35   {
36        bcolor = b;
37   }
38
39   void list_layout_node::setstr(const char * s)
40   {
41        strcpy(str, s);
42   }
43
44   void list_layout_node::setprint_type(int p)
45   {
46        print_type = p;
47   }
48
49   //Getters
50   list_layout_node * list_layout_node::getnext()
51   {
52        return next;
53   }
54
55   coord list_layout_node::getpos()
56   {
57        return pos;
58   }
59
60   int list_layout_node::gettcolor()
61   {
62        return tcolor;
63   }
64
65   int list_layout_node::getbcolor()
66   {
67        return bcolor;
```

```
68  }

70  const char * list_layout_node::getstr()
71  {
72      return str;
73  }

75  int list_layout_node::getprint_type()
76  {
77      return print_type;
78  }

80  void list_layout::print(int print_mode)
81  {
82      coord init_pos(wherex(), wherey());
83      for(list_layout_node *curr = head; curr; curr = curr->getnext())
84      {
85          coord c = curr->getpos();
86          int new_y = c.y - lines_scrolled;

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

91          gotoxy(c.x, new_y);
92          textcolor(curr->gettcolor());
93          textbackground(curr->getbcolor());
94          if(print_mode == DISPLAY)
95          {
96              if(curr->getprint_type() ==
97                  list_layout_node::PASSWORD)
98              {
99                  int len = strlen(curr->getstr());
100                 for(int i = 0; i < len; i++)
101                 {
102                     cprintf("*");
103                 }
104             }
105             else if(current->getprint_type() ==
106                     list_layout_node::DEFAULT)
107             {
108                 cprintf("%s", curr->getstr());
109             }
110         }
111         else if(print_mode == HIDE)
112         {
113             int len = strlen(curr->getstr());
114             for(int i = 0; i < len; i++)
115             {
116                 cprintf(" ");
117             }
118         }
119     }
120     gotoxy(init_pos.x, init_pos.y);
121 }

123 list_layout::list_layout()
124 {
125     head = NULL,
126     current = NULL;
```

129

```cpp
127
128        tcolor = ui::tcolor;
129        bcolor = ui::bcolor;
130        tcolor_selected = ui::bcolor;
131        bcolor_selected = ui::tcolor;
132        tcolor_input = tcolor;
133        bcolor_input = bcolor;
134
135        height = ui::scr_height − 1;
136        width = ui::scr_width;
137        lines_scrolled = 0;
138    }
139
140    list_layout& list_layout::operator<<(coord c)
141    {
142        pos = c;
143        return *this;
144    }
145
146    list_layout& list_layout::operator<<(const char *str)
147    {
148        if(!head) //empty list
149        {
150            head = new list_layout_node;
151            current = head;
152        }
153        else
154        {
155            list_layout_node *new_node = new list_layout_node;
156            current−>setnext(new_node);
157            current = current−>getnext();
158        }
159
160        current−>setpos(pos);
161        current−>setstr(str);
162        current−>settcolor(tcolor);
163        current−>setbcolor(bcolor);
164
165        print();
166
167        return *this;
168    }
169
170    interactive * list_layout::settext_box(coord c, int is_pwd)
171    {
172        interactive *new_node = new text_box;
173        new_node−>setpos(c);
174        new_node−>settcolor(tcolor_input);
175        new_node−>setbcolor(bcolor_input);
176
177        if(is_pwd)
178        {
179            ((text_box *) new_node)−>setis_password(1);
180            new_node−>setprint_type(list_layout_node::PASSWORD);
181        }
182
183        current−>setnext(new_node);
184        current = current−>getnext();
185
```

```
186      return new_node;
187  }
188
189  interactive * list_layout::setbutton(coord c, const char *s)
190  {
191      button *new_node = new button;
192      new_node->setpos(c);
193      new_node->settcolor(tcolor);
194      new_node->setbcolor(bcolor);
195      new_node->settcolor_selected(tcolor_selected);
196      new_node->setbcolor_selected(bcolor_selected);
197      new_node->setstr(s);
198
199      interactive *n = (interactive *) new_node;
200      current->setnext(n);
201      current = current->getnext();
202
203      return n;
204  }
205
206  void list_layout::settcolor(int c)
207  {
208      tcolor = c;
209      tcolor_input = c;
210  }
211
212  void list_layout::setbcolor(int c)
213  {
214      bcolor = c;
215      bcolor_input = c;
216  }
217
218  void list_layout::settcolor_selected(int c)
219  {
220      tcolor_selected = c;
221  }
222
223  void list_layout::setbcolor_selected(int c)
224  {
225      bcolor_selected = c;
226  }
227
228  void list_layout::settcolor_input(int c)
229  {
230      tcolor_input = c;
231  }
232
233  void list_layout::setbcolor_input(int c)
234  {
235      bcolor_input = c;
236  }
237
238  void list_layout::setcorner_top_left(coord c)
239  {
240      hide();
241
242      coord offset = c - corner_top_left;
243      //offset isn't a coordinate but it's just a pair of values
244
```

```
245         for(list_layout_node *curr = head; curr; curr = curr->getnext())
246         {
247             coord a = curr->getpos();
248             a += offset;
249             curr->setpos(a);
250         }
251
252         corner_top_left += offset;
253         pos += offset;
254
255         display();
256     }
257
258     void list_layout::setheight(int h)
259     {
260         hide();
261         height = h;
262         display();
263     }
264
265     void list_layout::setwidth(int w)
266     {
267         width = w;
268     }
269
270     void list_layout::setlines_scrolled(int l)
271     {
272         hide();
273         lines_scrolled = l;
274         display();
275     }
276
277     void list_layout::setpos(coord c)
278     {
279         pos = c;
280     }
281
282     int list_layout::getheight()
283     {
284         return height;
285     }
286
287     int list_layout::getwidth()
288     {
289         return width;
290     }
291
292     int list_layout::getlines_scrolled()
293     {
294         return lines_scrolled;
295     }
296
297     coord list_layout::getpos()
298     {
299         return pos;
300     }
301
302     coord list_layout::getcorner_top_left()
303     {
```

```
304        return corner_top_left;
305    }
306
307    void list_layout::display()
308    {
309        print(DISPLAY);
310    }
311
312    void list_layout::hide()
313    {
314        print(HIDE);
315    }
316
317    void list_layout::clear()
318    {
319        list_layout_node *curr = head;
320        head = current = NULL;
321
322        while(curr)
323        {
324            list_layout_node *temp = curr->getnext();
325            delete curr;
326            curr = temp;
327        }
328
329        lines_scrolled = 0;
330        pos = corner_top_left;
331    }
```

17. code/UI/button.cpp

```
1    #include "ui/ui.hpp"
2
3    button::button()
4    {
5        tcolor_selected = BLACK;
6        bcolor_selected = LIGHTGRAY;
7    }
8
9    void button::settcolor_selected(int c)
10   {
11       tcolor_selected = c;
12   }
13
14   void button::setbcolor_selected(int c)
15   {
16       bcolor_selected = c;
17   }
18
19   int button::gettcolor_selected()
20   {
21       return tcolor_selected;
22   }
23
24   int button::getbcolor_selected()
25   {
26       return bcolor_selected;
27   }
```

```
28
29  int button::input(int offset)
30  {
31      coord c = getpos();
32      setoffset(offset);
33      c.y += offset;
34      gotoxy(c.x, c.y);
35
36      print(1);
37
38      int state_to_return;
39      while(1)
40      {
41          if(kbhit())
42          {
43              char ch = interactive::getkey();
44              switch((int) ch)
45              {
46                  case interactive::ENTER :
47                      state_to_return = interactive::CLICKED;
48                      goto next;
49                  case interactive::DOWN :
50                  case interactive::TAB   :
51                      state_to_return = interactive::GOTONEXT;
52                      goto next;
53                  case interactive::UP :
54                  case interactive::SHIFT_TAB :
55                      state_to_return = interactive::GOTOPREV;
56                      goto next;
57                  case interactive::SHIFT_BACKSPACE :
58                      state_to_return = interactive::BACK;
59                      goto next;
60              }
61          }
62      }
63
64      next:
65      {
66          if (
67              state_to_return == interactive::GOTONEXT ||
68              state_to_return == interactive::GOTOPREV
69             )
70          {
71              print(0);
72          }
73
74          return state_to_return;
75      }
76  }
77
78  void button::print(int isselected)
79  {
80      if(isselected)
81      {
82          textcolor(tcolor_selected);
83          textbackground(bcolor_selected);
84      }
85      else
86      {
```

134

```
87          textcolor(gettcolor());
88          textbackground(getbcolor());
89      }
90
91      coord init_pos(wherex(), wherey());
92      coord c = getpos();
93      gotoxy(c.x, c.y + getoffset());
94      cprintf(getstr());
95      gotoxy(init_pos.x, init_pos.y);
96  }
```

18. code/UI/textbox.cpp

```
1   #include "ui/ui.hpp"
2
3   text_box::text_box()
4   {
5       is_password = 0;
6   }
7
8   /*
9    * Despite trying, this function has grown quite large
10   * Basically, it allows the user to enter text in the box
11   * and stores it.
12   * Returns GOTONEXT or GOTOPREV as per user's request to
13   * go to the next or the previous text box respectively
14   */
15  int text_box::input(int a)
16  {
17      coord c = getpos();
18      setoffset(a);
19      c.y += a;
20      gotoxy(c.x, c.y);
21
22      const char *string = getstr();
23      char str[100];
24      strcpy(str, string);
25
26      string_node *head = new string_node,
27                  *current = head;
28
29      int len = strlen(str);
30      string_node *temp_prev = NULL;
31      for(int i = 0; i < len ; i++)
32      {
33          current->data = str[i];
34          current->next = new string_node;
35          current->prev = temp_prev;
36          temp_prev = current;
37          current = current->next;
38      }
39
40      //At the end is a box with \0
41      current->data = '\0';
42      current->prev = temp_prev;
43      current = head;
44
45      int state_to_return = -1;
```

135

```
46
47     while(1)
48     {
49         if(kbhit())
50         {
51             char ch = interactive::getkey();
52
53             switch((int)ch)
54             {
55                 case TAB :
56                 case ENTER :
57                     state_to_return = GOTONEXT;
58                     goto convert_to_str;
59                 case BACKSPACE :
60                     if(current)
61                     {
62                         if(!current->prev)  break; //No character to be deleted
63
64                         string_node *node_to_delete = current->prev;
65
66                         if(node_to_delete->prev) node_to_delete->prev->next =
                                current;
67                         else                      head = current; //If the node to
                                be deleted is the head
68
69                         current->prev = node_to_delete->prev;
70
71                         delete node_to_delete;
72
73                         gotoxy(wherex() - 1, wherey());
74
75                         print_str(head);
76                     }
77                     break;
78                 case DELETE:
79                     if(current)
80                     {
81                         if(current->data == '\0') break; //No character to be
                                deleted
82
83                         string_node *node_to_delete = current;
84
85                         if(current->prev) current->prev->next = current->next;
86                         else              head = current->next;
87
88                         if(current->next) current->next->prev = current->prev;
89
90                         current = current->next;
91                         delete node_to_delete;
92
93                         print_str(head);
94
95                     }
96                     break;
97                 case HOME:
98                     gotoxy(c.x, c.y);
99                     current = head;
100                    break;
101                case END:
```

136

```
102                     while(current->next)
103                     {
104                         current = current->next;
105                         gotoxy(wherex()+1, wherey());
106                     }
107                     break;
108                 case SHIFT_BACKSPACE:
109                     state_to_return = BACK;
110                     goto convert_to_str;
111                 case SHIFT_TAB:
112                     state_to_return = GOTOPREV;
113                     goto convert_to_str;
114                 case UP:
115                     state_to_return = GOTOPREV;
116                     goto convert_to_str;
117                 case DOWN:
118                     state_to_return = GOTONEXT;
119                     goto convert_to_str;
120                 case LEFT:
121                     if(current->prev)
122                     {
123                         current = current->prev;
124                         gotoxy(wherex()-1, wherey());
125                     }
126                     break;
127                 case RIGHT: //Right arrow key
128                     if(current->next)
129                     {
130                         current = current->next;
131                         gotoxy(wherex()+1, wherey());
132                     }
133                     break;
134                 default:
135                     if(isprint(ch))
136                     {
137                         /*
138                          * When a new node is to be added, it is added behind
139                          * the current node
140                          */
141
142                         string_node *new_node = new string_node;
143                         new_node->data = ch;
144                         new_node->next = current;
145                         new_node->prev = current->prev;
146
147                         if(current->prev) current->prev->next = new_node;
148                         else              head = new_node;
149                         current->prev = new_node;
150
151                         gotoxy(wherex()+1, wherey());
152
153                         print_str(head);
154                     }
155             }
156         }
157     }
158
159     convert_to_str:
160     {
```

```
161          char a[100]; int insert_pointer = 0;
162          for(current = head; current; current = current->next)
163          {
164              a[insert_pointer] = current->data;
165              insert_pointer++;
166          }
167
168          setstr(a);
169
170          //Deleting the list
171          current = head;
172          head = NULL;
173          while(current)
174          {
175              string_node *temp = current->next;
176              delete current;
177              current = temp;
178          }
179
180          return state_to_return;
181      }
182
183  }
184
185  /*
186   * Prints the string as represented by a doubly
187   * linked list whose head is pointed to by the
188   * parameter.
189   */
190  void text_box::print_str(string_node *head)
191  {
192      coord init = coord(wherex(), wherey());
193      coord c = getpos();
194      gotoxy(c.x, c.y + getoffset());
195      textcolor(gettcolor());
196      textbackground(getbcolor());
197      for(string_node *current = head; current; current = current->next)
198      {
199          if(is_password)
200          {
201              if(current->data != '\0')
202              {
203                  cprintf("*");
204              }
205              else
206              {
207                  cprintf(" ");
208              }
209          }
210          else            cprintf("%c", current->data);
211      }
212      gotoxy(init.x, init.y);
213  }
214
215  void text_box::setis_password(int a)
216  {
217      is_password = a;
218  }
```

19. code/UI/infotbox.cpp

```cpp
1  #include "ui/ui.hpp"
2  #include "iface.hpp"
3
4  info_tbox::info_tbox()
5  {
6      tbox = NULL;
7      data_store = NULL;
8      type = OTHER;
9      validator = NULL;
10 }
11
12 int info_tbox::setdata()
13 {
14     if(validator(tbox->getstr()) == 0)
15     {
16         return 0;
17     }
18
19     char *fstr;
20     switch(type)
21     {
22         case INT:
23         {
24             fstr = "%d";
25             break;
26         }
27         case LONG:
28         {
29             fstr = "%ld";
30             break;
31         }
32         case UNSIGNED_LONG:
33         {
34             fstr = "%lu";
35             break;
36         }
37         case STRING:
38         case PASSWORD:
39         {
40             char *s = (char *) data_store;
41             strcpy(s, tbox->getstr());
42             return 1;
43         }
44         case CHAR:
45         {
46             fstr = "%c";
47             break;
48         }
49         case DOUBLE:
50         {
51             fstr = "%g";
52             break;
53         }
54         case FLOAT:
55         {
56             fstr = "%f";
57             break;
```

```
58          }
59       default:
60          return 0;
61    }
62
63    sscanf(tbox->getstr(), fstr, data_store);
64
65    return 1;
66 }
```

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

LHOSPITAL

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

Choice : 1

Everything looks OK

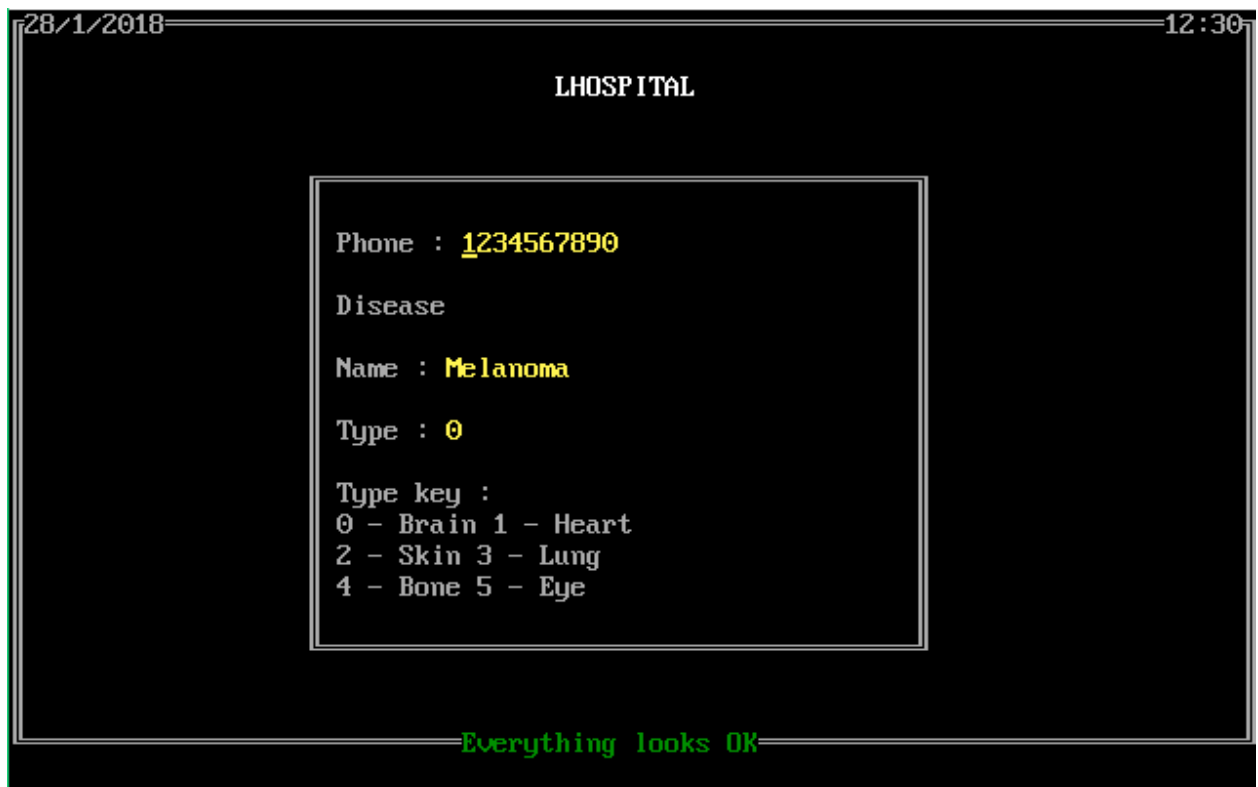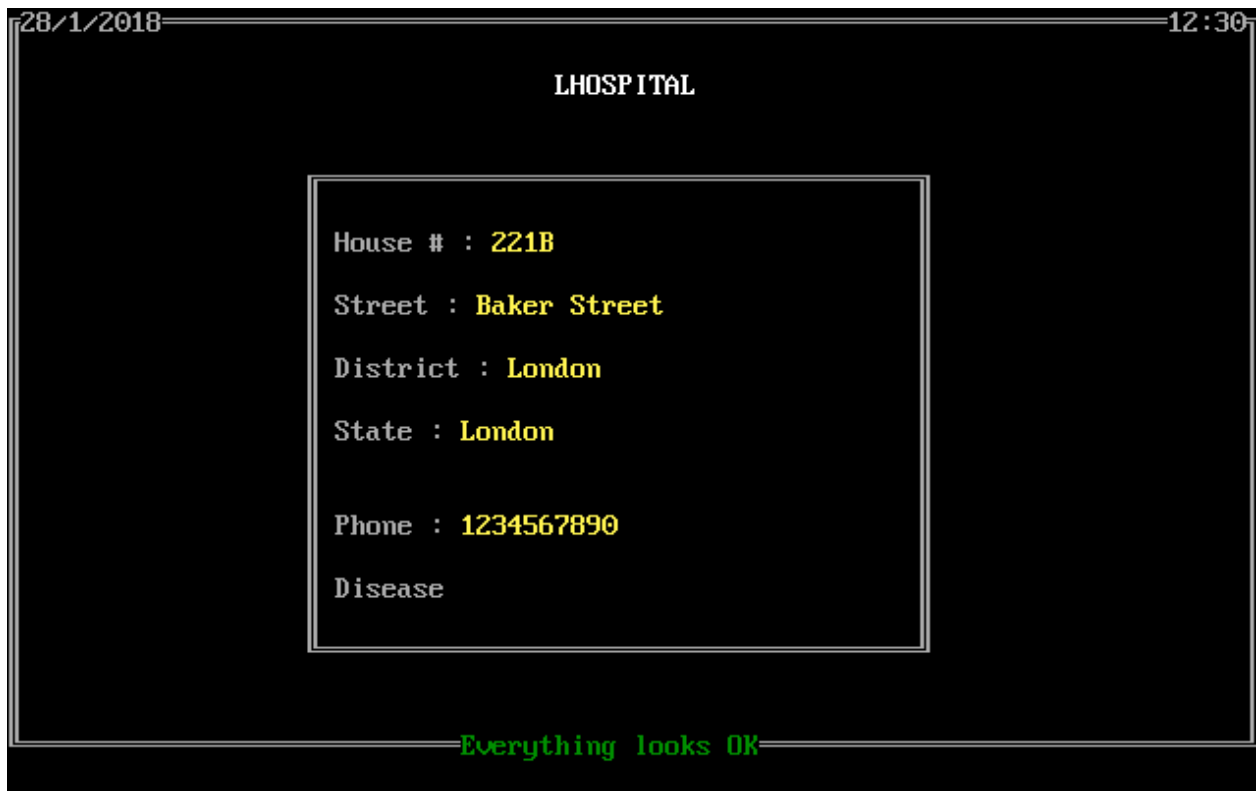LHOSPITAL

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

LHOSPITAL

House # : 221B

Street : Baker Street

District : London

State : London

Phone : 1234567890

Disease

Everything looks OK

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

LHOSPITAL

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

Symptom 2 :

Symptom 3 :

Symptom 4 :


Guardian name : Dr. John Watson
```

Everything looks OK

LHOSPITAL

```
Patient has been
admitted with ID # 14

       Okay
```

Everything looks OK

LHOSPITAL

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

Choice : 1

Everything looks OK

LHOSPITAL

Medicine Sale
Code : 1

Submit

Everything looks OK

LHOSPITAL

```
          Medicine Sale
    Name : LZMA
    Price : $ 18.990000

    Patient ID : 14

    Quantity : 15
              Submit
```

LHOSPITAL

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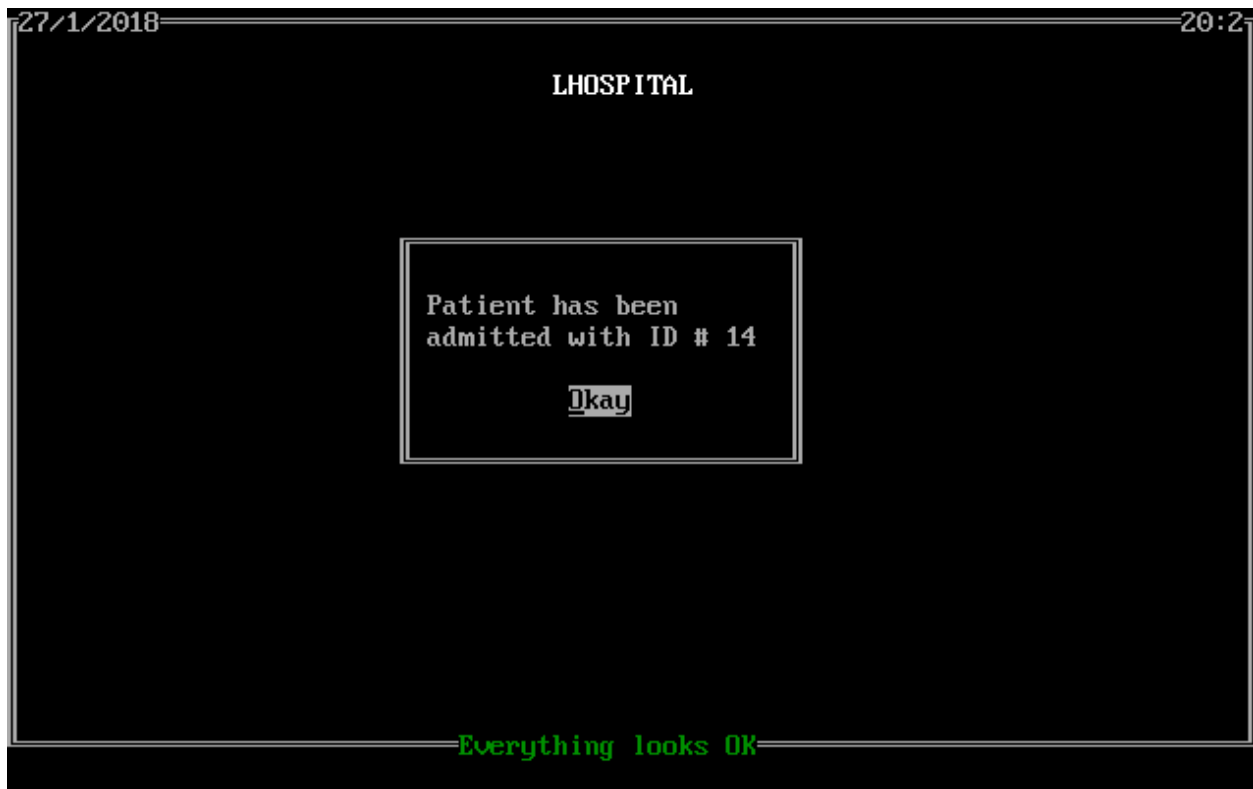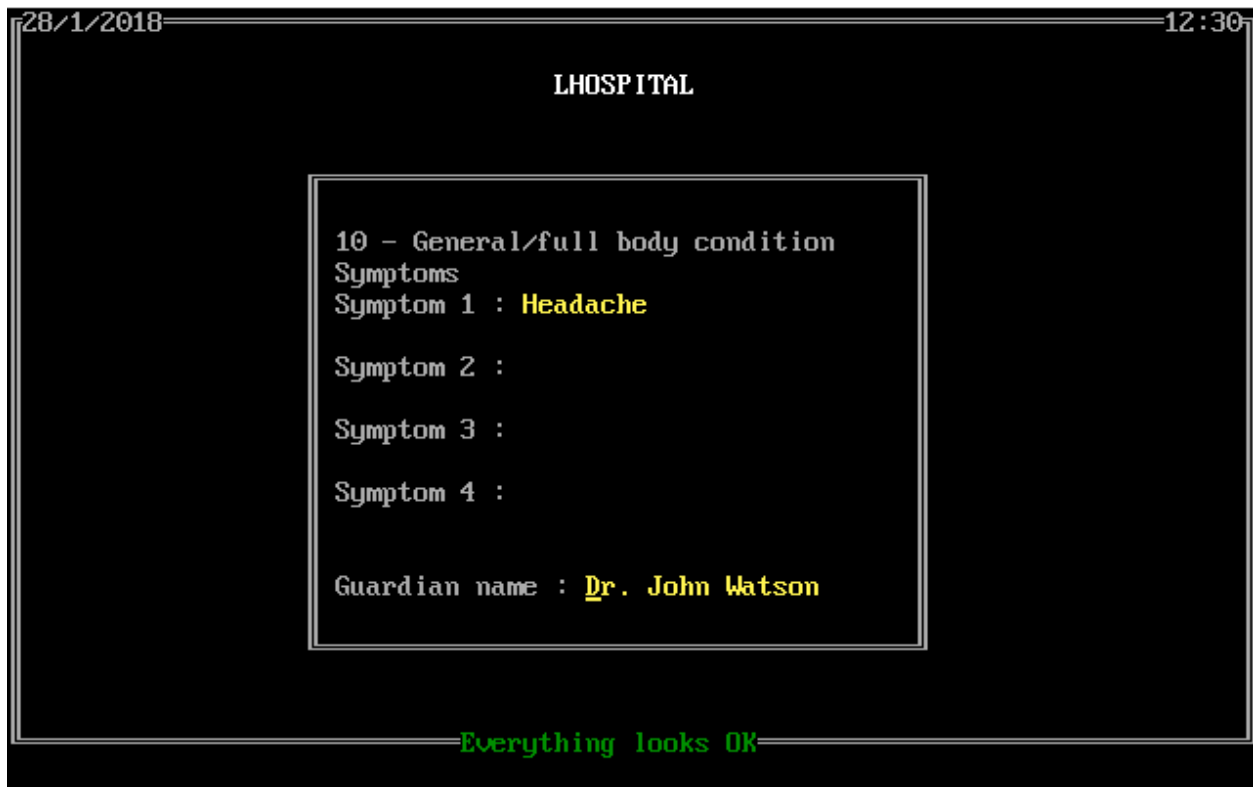
LHOSPITAL

Patient Data
Alteration
Enter patient ID : 14

Submit

LHOSPITAL

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

Choice : 1

Submit

LHOSPITAL

Enter insurance
information for Khan
Khaneja
Provider : **LIC**

Amount (in $) : **30000**

Expiry date

Everything looks OK

LHOSPITAL

Patient Discharge
Enter patient ID : 14

Submit

Everything looks OK

LHOSPITAL

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

Everything looks OK