Contents

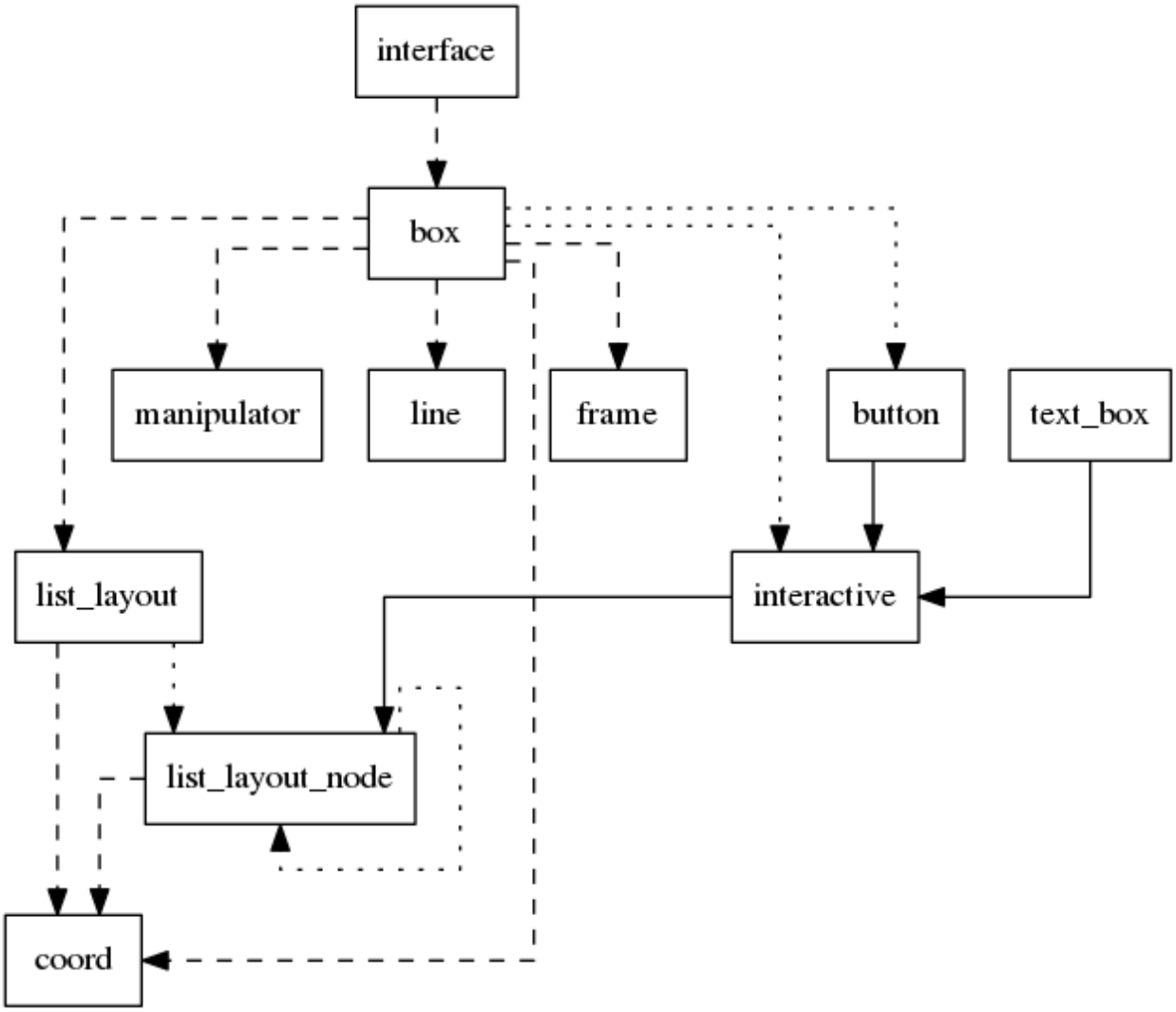
1 Diagrams **2**
1.1 ER diagrams 2
1.2 Flowchart of main() 4

2 Source Code **5**
2.1 Header files 5
2.2 C++ files (.cpp) 5
2.3 Data files 6

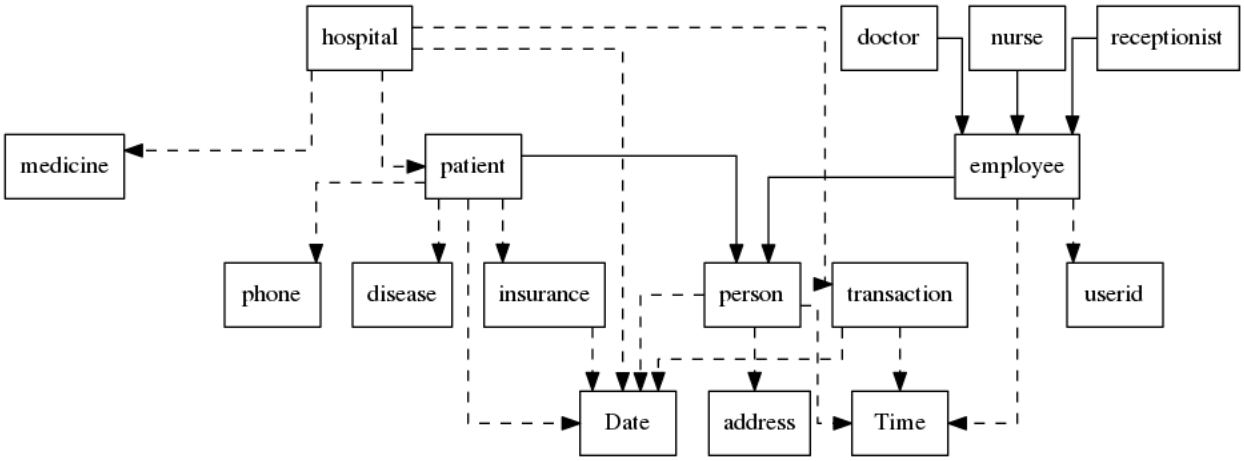
Diagrams

ER diagrams

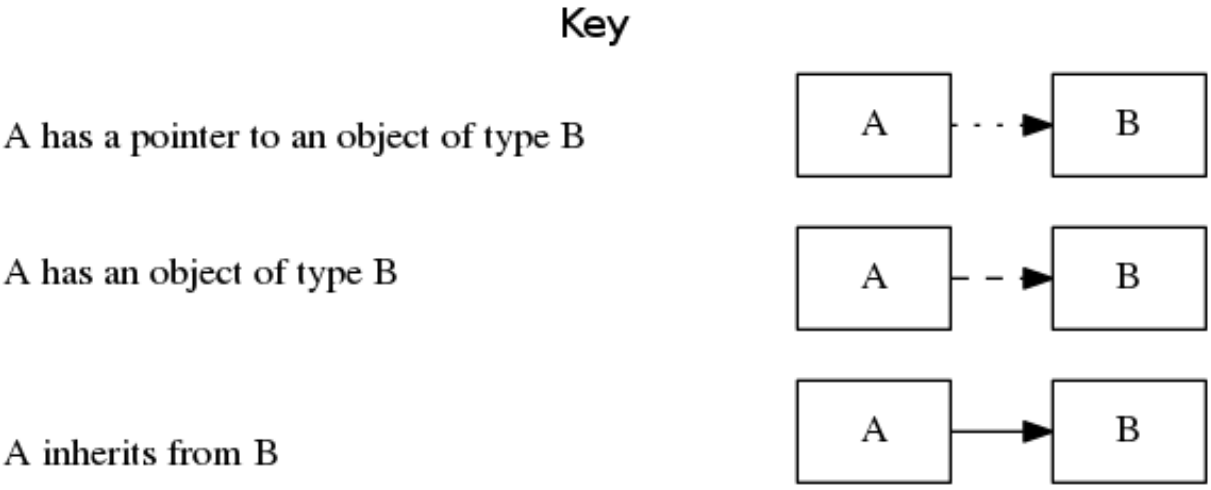
1. User interface



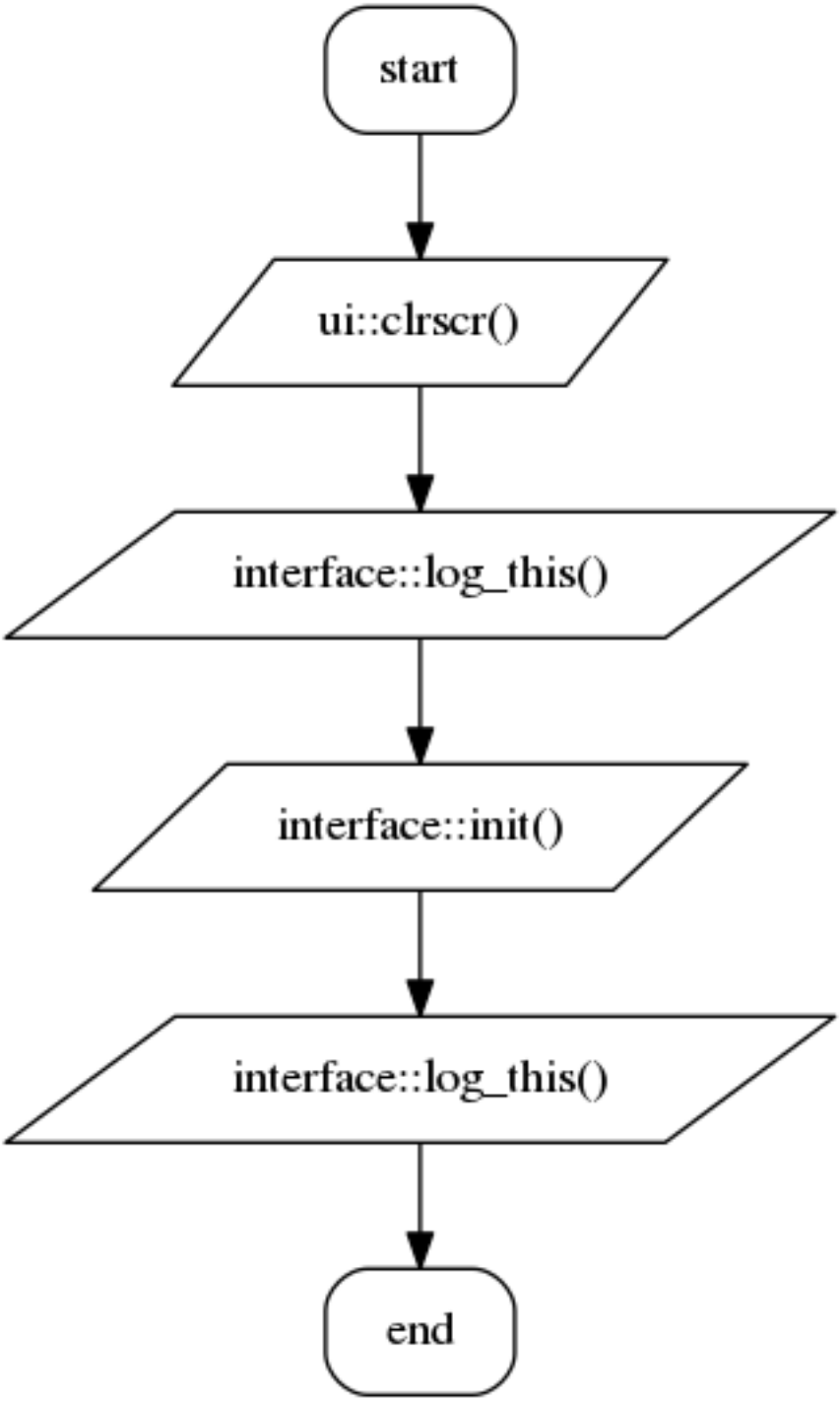
2. Hospital



3. Key to ER diagrams



Flowchart of main()



Source Code

Header files

1. code/iface.hpp

```
#ifndef INTERFACE_HPP
#define INTERFACE_HPP

#include "ui/ui.hpp"

class interface{
    public:
        static void init();
        static int login_screen();
        static int menu();
        static void patient_management();
        static void employee_management();
        static void stock_management();

        static void doctor_screen();
        static void nurse_screen();
        static void receptionist_screen();

        static void error(char*);
        static void clear_error();

        static int log_this(char *);

    private:
        interface();
        class validate_menu         //for creating a validation function to use
            in menus
        {                           //to validate the choice input of the menu
            option to be accessed
                static int lowest_choice, greatest_choice;
                validate_menu();
            public:
                static int input(const char *);
                static void set_menu_limits(int, int);
        };
        static box window;
};

#endif /* INTERFACE_HPP */
```

2. code/EMP.HPP

```
#ifndef EMP
#define EMP

#include "base.hpp"

enum emp_type {INVALID, OTHERS, DOCTOR, NURSE, RECEPTIONIST};

class employee : public person{
        int generate_id();
```

```
10          static int generate_id_status;
11      public:
12          employee(str, int, Date, address, phone, unsigned long, Time, Time, str =
                ", str = "");    //for all those with user accounts(doctors, nurses,
                receptionists), last 2 arguments are to be provided as well
13          employee(); //default constructor
14          int get_age();      //overridden function
15          unsigned long get_salary();
16          void set_salary(unsigned long);
17          Time get_shift(int);
18          void set_shift(int, Time);
19          unsigned long get_id();
20          transaction * get_last_10_transactions();
21          static int get_generate_id_status();
22          userid account;
23      protected:
24          unsigned long id;
25          unsigned long salary;
26          Time shift_start;
27          Time shift_end;
28  };
29
30  class doctor : public employee{
31      public:
32          doctor(str, int, Date, address, phone, unsigned long, Time, Time, int,
                int, str, str);
33          doctor();   //default constructor
34          int * get_speciality();
35          long * get_patients();
36      private:
37          int speciality[2];              // Doctor's specialization
38          long patients[10];              // Patients currently under care, can
                take only 10 at once
39  };
40
41  class nurse : public employee{
42      public:
43          nurse(str, int, Date, address, phone, unsigned long, Time, Time, str, str
                );
44          nurse();        //default constructor
45          long * get_patients();
46      private:
47          long patients[5];
48  };
49
50  class receptionist : public employee
51  {
52      public:
53          receptionist(str, int, Date, address, phone, unsigned long, Time, Time,
                str, str);
54          receptionist();
55  //      doctor assign_doctor(patient);
56  };
57
58  class id_to_emp
59  {
60          unsigned long id;
61          int employee_type;
62      public:
```

```
63          int status; //true whenever the constructor runs successfully and
                succeeds in storing the object to id_list.dat
64          id_to_emp(unsigned long, int);
65          id_to_emp();
66          static int convert(unsigned long);
67  };
68
69  #endif
```

3. code/BASE.HPP

```
1   #ifndef BASE
2   #define BASE
3
4   #include "ui/ui.hpp"
5   #include <fstream.h>
6   #include <string.h>
7   #include <dir.h>
8   #include <stdio.h>
9   #include <math.h>
10  #include <string.h>
11  #include <time.h>
12  #include <stdlib.h>         //for random() and randomize()
13
14  const int K = 14;
15  typedef char str[80];
16  typedef char phone[11];
17
18  enum sex {MALE, FEMALE, TRANS};
19  enum date_type {DAY, MONTH, YEAR};
20  enum time_type {HOUR, MINUTE, SECOND};
21  enum body_parts {BRAIN, HEART, SKIN,
22      LUNG, BONE, EYE,
23      THROAT, TEETH, STOMACH,
24      BLOOD, GUT, GEN};   // GEN for general problems
25  enum address_parts {HOUSE_NO, STREET, CITY, DISTRICT, STATE};
26  enum times_of {START, END};
27
28  struct Time{
29      unsigned int hour;
30      unsigned int minute;
31      unsigned int second;
32
33      Time();
34      Time(unsigned h, unsigned m, unsigned s);
35  };
36
37  struct Date{
38      unsigned int day;
39      unsigned int month;
40      unsigned int year;
41
42      Date();
43      Date(unsigned d, unsigned m, unsigned y);
44  };
45
46  class system
47  {
48      private:
```

```
49          system();
50      public:
51          static Date get_date();
52          static Time get_time();
53  };
54
55  struct address{
56      str house_no;
57      str street;
58      str city;
59      str district;
60      str state;
61  };
62
63  struct disease{
64      str name;
65      int type;           //refers to body part affected (LUNG, HEART, etc)
66      str symptoms[4];    //symptoms reported by patient
67  };
68
69  struct insurance{
70      str provider;
71      unsigned long amount;
72      Date expiry;
73  };
74
75  struct medicine{
76      int code;
77      float price;
78      str name;
79      float dosage;
80      long stock;
81  };
82
83  struct transaction{
84      float amount;
85      str reason;
86      Date _date;
87      Time _time;
88      transaction(float, Date = Date(), Time = Time(), char* = "NA");
89      transaction();
90  };
91
92  struct procedure{
93      str name;
94      float cost;
95  };
96
97  class person{
98      public:
99          person(str, int, Date, address, phone); // Explicit constructor
100         person();
101         // 'Get's
102         char* get_name();
103         int get_age();
104         int get_sex();
105         Date get_dob();
106         address get_address();
107         char* get_phone();
```

```
108
109         //Updating functions
110         void set_name(char*);
111         void set_sex(int);
112         void set_dob(Date, Date = system::get_date());
113         void set_address(address);
114         void set_phone(char*);
115
116     protected:
117         str name;
118         unsigned age;
119         unsigned sex;
120         Date dob;
121         address adr;
122         phone phone_no;
123
124     private:
125         void calc_age(Date = system::get_date());
126 };
127
128 class userid
129 {
130         str username;
131         str passcipher;             //encrypted password
132         str default_key;            //key for making the vigenere cipher
133         void makecipher(char *);    //makes the vigenere cipher
134         void set_key(char *);       //sets default_key
135         char * decipher();          //deciphers the cipher 'passcipher'
136
137     public:
138         userid(char *, char *);
139         userid();   //default constructor;
140         char * get_username();
141         void set_username(char *);
142         int login(char *);
143 };
144
145 class enum_to_str           //defines << operator overloads to facilitate
        printing of some stuff
146 {
147         enum_to_str();
148     public:
149         friend box & operator<<(box &output, sex s);        //converts sex
                enumeration constant into a string and prints it to a box
150         friend box & operator<<(box &output, body_parts b); //converts body_parts
                 enumeration constant into a string and prints it to a box
151         friend box & operator<<(box &output, Time & t);     //converts Time
                variable into a string and prints it to a box
152         friend box & operator<<(box &output, Date & d);     //converts Date
                variable into a string and prints it to a box
153         friend box & operator<<(box &output, address & a);  ////converts address
                variable into a string and prints it to a box
154 };
155
156 #endif
```

4. code/PATIENT.HPP

```
1 #ifndef PATIENT
```

```
#define PATIENT

#include "base.hpp"

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

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

#endif
```

5. code/HOSP.HPP

```
#ifndef HOSP
#define HOSP

#include "base.hpp"
```

```
5  #include "patient.hpp"
6
7  const int monthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
8  const int stay_charge = 50;  // $50 per day
9
10 class hospital
11 {
12     public:
13         static float get_bal();
14         static transaction deduct_money(float, char*, Date, Time);
15         static transaction add_money(float, char*, Date, Time);
16         static transaction* get_transaction();
17         static void read_balance();
18
19         static patient get_patient_by_id(int);
20         static void write_patient(patient);
21         static void charge_patient(int, transaction);
22         static void discharge_patient(patient);
23         static float calc_bill(int);
24
25         static medicine get_med_by_code(int);
26
27         static int get_employee_by_id(unsigned long, void *);
28 //      static int get_employee_by_id(unsigned long, doctor &); //The new
       get_employee_by_id hasn't been tested properly, so until that is done, these
       functions are gonna remain commented
29 //      static int get_employee_by_id(unsigned long, nurse &);
30 //      static int get_employee_by_id(unsigned long, receptionist &);
31         static int write_employee(void *);
32 //      static int write_employee(doctor);                    //same as above
       for write function
33 //      static int write_employee(nurse);
34 //      static int write_employee(receptionist);
35         static int pay_salary(unsigned long, Date, Time);
36         static int pay_all_salaries();
37
38         static int get_date_difference(Date, Date);
39         static int count_leap_years(Date);
40         static int date_validity(const char *);
41         static int date_validity(Date);
42         static Date str_to_date(const char *);
43         static int str_to_sex(char *);
44
45     private:
46         hospital();
47         static int read_from(unsigned long, char *, int, char *);
48         static double balance;
49 };
50
51 class pharmacy{
52     public:
53         static void sale(int, int, int);
54         static void purchase(int, int);
55 };
56
57 #endif
```

6. code/UI/test.hpp

```
#ifndef TEST_HPP
#define TEST_HPP

void test_weird_error();

int back_func();
void test_back();

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

#endif /* TEST_HPP */
```

7. code/UI/ui.hpp

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

#ifndef UI_HPP
#define UI_HPP

#include <conio.h>
#include <stdarg.h>
#include <string.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
#include <stdlib.h>
#include <limits.h>
#include <errno.h>

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

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

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

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

```
43 {
44     static int index; //!< index of a new manipulator object
45     int own_index;    //!< index of current manipulator
46 
47     public:
48         manipulator(); //!< Ctor; assigns index
49         int operator==(manipulator); //!< Returns 1 if indexes are same
50 };
51 
52 //! Class containing basic UI functions and attributes
53 class ui
54 {
55     ui();   //!< Private ctor; object of this class shouldn't be created
56     public:
57 
58         //! Specifies the directions for modifying frame, etc.
59         enum dir
60         {
61             left = 1,
62             top = 2,
63             right = 4,
64             bottom = 8,
65             all = 16 //!< When all sides need to be modified
66         };
67         static int scr_height; //!< Height of screen
68         static int scr_width; //!< Width of screen
69         static void init(); //!< Sets all static variables
70         static void clrscr(); //!< Clears the contents off the screen
71         static int tcolor; //!< text color
72         static int bcolor; //!< background color
73         static manipulator endl; //!< End line and move cursor to next line
74         static manipulator centeralign; //!< Center align
75         static manipulator rightalign; //!< Right align
76 };
77 
78 //! Represents a coordinate
79 struct coord
80 {
81     int x;  //!< x coordinate
82     int y;  //!< y coordinate
83 
84     coord(int = 1,int = 1); //!< Sets the coordinate
85     coord & operator+=(coord);
86     coord & operator-=(coord);
87     coord operator+(coord);
88     coord operator-(coord);
89 };
90 
91 //! Represents the node of a list representing the layout
92 /*!
93  Represents all the information of an element that will be
94  printed on the screen. Also points to the next element of the
95  screen that will be printed next to it
96 */
97 class list_layout_node
98 {
99     list_layout_node *next;     //!< Pointer to next node
100    coord pos;                  //!< Position where to print
101    int tcolor;                 //!< Text colour
```

```
102     int bcolor;                 //!< Background colour
103     char str[100];              //!< String to print
104 
105     //! How to print the string; mainly for passwords
106     int print_type;
107 
108     public:
109         list_layout_node();     //!< Ctor
110         ~list_layout_node();    //!< Dtor
111 
112         //!@{ Setter functions
113         void setnext(list_layout_node *);
114         void setpos(coord);
115         void settcolor(int);
116         void setbcolor(int);
117         void setstr(const char *);
118         void setprint_type(int);
119         //!@}
120 
121         //!@{ Getter functions
122         list_layout_node * getnext();
123         coord getpos();
124         int gettcolor();
125         int getbcolor();
126         const char * getstr();
127         int getprint_type();
128         //!@}
129 
130         //! Used to distinguish will be printed i.e.
131         //! as is or hidden (as passwords)
132         enum print_types
133         {
134             DEFAULT,
135             PASSWORD
136         };
137 };
138 
139 //! A node of the representation of string as a linked list
140 struct string_node
141 {
142     string_node *next;  //!< Pointer to next node
143     string_node *prev;  //!< Pointer to previous node
144     char data;          //!< Character stored in string
145 
146     string_node();      //!< Ctor
147 };
148 
149 //! Represents all interactive information
150 /*!
151  Basically a parent class of all the classes that
152  represent the elements of the layout the user can
153  interact with.
154  Used so that all those elements can be clubbed together
155  and the input be taken.
156 */
157 class interactive : public list_layout_node
158 {
159     interactive *prev;      //!< ptr to previous node
160     interactive *next;      //!< ptr to next node
```

```
161     int offset;             //!< offset to y position when printing
162     public:
163         interactive();      //!< Ctor
164         ~interactive();     //!< Dtor
165 
166         //! Empty input function that will be overridden by children
167         /*!
168          \param offset The offset to y position
169          \return Action that was performed by the user
170         */
171         virtual int input(int offset);
172 
173         //! Setter function
174         void setoffset(int);
175 
176         //! Getter function
177         int getoffset();
178 
179         //! Actions that are performed by user; returned from input func.
180         enum actions
181         {
182             GOTONEXT,
183             GOTOPREV,
184             CLICKED,
185             BACK //!< When shift-bckspc is pressed
186         };
187 
188         //! Keys that user can press to navigate the form
189         enum keys
190         {
191             TAB,
192             ENTER,
193             BACKSPACE,
194             SHIFT_BACKSPACE,
195             SHIFT_TAB,
196             HOME,
197             END,
198             DELETE,
199             UP,
200             DOWN,
201             LEFT,
202             RIGHT
203         };
204 
205         //! Gets key from user and returns code
206         /*
207          \return Keyname corresponding to enum keys
208         */
209         static int getkey();
210 };
211 
212 //! Represents a text box
213 /*!
214  Inherits from interactive as a text box can be interacted
215  with. Gets data from user and stores it as a string that
216  can be further converted to the required data type
217 */
218 class text_box : public interactive
219 {
```

```
220     //! Represents if the data entered in the text box
221     //! should be displayed as is or replaced with asterisks
222     int is_password;
223 
224     public:
225         text_box(); //!< Ctor
226 
227         //! Takes input and returns user action
228         /*!
229          /param offset Offset of y coordinate to print
230          /return Action performed by user
231         */
232         int input(int offset = 0);
233 
234         //! Prints string represented by a linked list
235         /*
236          Takes in the head pointer of the linked list
237          string and prints the string by iterating through
238          the list. Has no other side effects.
239          /param head ptr to head of the linked list
240         */
241         void print_str(string_node *head);
242 
243         //! Setter function
244         void setis_password(int);
245 };
246 
247 //! Represents a button that can be clicked
248 /*!
249  Inherits from interactive as a button can be interacted with.
250  A user can click the button while it's input function is
251  running which will return the user action
252 */
253 class button : public interactive
254 {
255     int tcolor_selected; //!< tcolor when selected
256     int bcolor_selected; //!< bcolor when seilected
257 
258     public:
259         button(); //!< Ctor
260 
261         //!@{ Setter functions
262         void settcolor_selected(int);
263         void setbcolor_selected(int);
264         //!@}
265 
266         //!@{ Getter functions
267         int gettcolor_selected();
268         int getbcolor_selected();
269         //!@}
270 
271         //! Input function
272         /*!
273          Effectively allows the button to be clicked
274          /param offset Offset of y coordinate to print
275          /return Action performed by the user
276         */
277         int input(int offset = 0);
278 
```

```
279         //! Prints the button
280         /*!
281          /param isselected Indicates if button is selected or not
282         */
283         void print(int isselected = 0);
284 };
285
286 //! Represents the layout of the page
287 /*!
288  Incorporates elements like simple nodes as well as other
289  interactive elements. This layout can be contained within
290  a specific height and the overflowing content can reached
291  by scrolling which is also implemented here.
292 */
293 class list_layout
294 {
295     //!@{ Pointers to implement a linked list to elements
296     list_layout_node *head; //!< ptr to head node
297     list_layout_node *current; //!< ptr to current node
298     //!@}
299
300     coord corner_top_left; //!< top left corner of container
301
302     /*!
303      Following are used as temporary placeholders till data
304      is written to the nodes
305     */
306     ///!@{
307     coord pos;
308     int tcolor;
309     int bcolor;
310     int tcolor_selected;
311     int bcolor_selected;
312     int tcolor_input;
313     int bcolor_input;
314     ///!@}
315
316     //!@{ For scrolling implementation
317     int height; //!< Height of the layout
318     int width; //!< Width of the layout
319     int lines_scrolled; //!< Lines currently scrolled
320     //!@}
321
322     //! For better verbosity at internal level
323     enum print_modes
324     {
325         DISPLAY,
326         HIDE
327     };
328
329     //! Prints the layout
330     /*!
331      Prints the layout by iterating through the internal
332      linked list maintained. Has no other side effects
333      /param print_mode How to print the data
334     */
335     void print(int print_mode = DISPLAY);
336     public:
337         list_layout(); //!< Ctor
```

```
338
339         //!@{ Set an element (node)
340         list_layout& operator<<(coord); //!< Set coord of node
341
342         //! Set data held by the node
343         list_layout& operator<<(const char *);
344         //!@}
345
346         //! Set a text box
347         /*!
348          Sets a text box at the position indicated by pos and
349          returns a pointer to it
350          /param pos Position at which to set text box
351          /param is_pass If the text box has a password, set to 1
352          /return pointer to the text box set (casted to interactive *)
353         */
354         interactive * settext_box(coord pos, int is_pass = 0);
355
356         //! Set a button
357         /*!
358          Sets a button at the position indicated by pos and
359          returns a pointer to it
360          /param pos Position at which to set the button
361          /param txt The text the button displays
362         */
363         interactive * setbutton(coord pos, const char *txt);
364
365         //!@{ Setter functions
366         void settcolor(int);
367         void setbcolor(int);
368         void settcolor_selected(int);
369         void setbcolor_selected(int);
370         void settcolor_input(int);
371         void setbcolor_input(int);
372         void setcorner_top_left(coord);
373         void setheight(int);
374         void setwidth(int);
375         void setlines_scrolled(int);
376         void setpos(coord);
377         //!@}
378
379         //!@{ Getter functions
380         int getheight();
381         int getwidth();
382         int getlines_scrolled();
383         coord getpos();
384         coord getcorner_top_left();
385         //!@}
386
387         void display(); //!< Display the layout
388         void hide(); //!< Hide the layout
389         void clear(); //!< Deletes contents of the layout
390 };
391
392 //! Represents a border
393 /*!
394  Basically represents a border with characters that can be
395  customised to suit the requirements.
396 */
```

```
397 class frame
398 {
399     char border_chars[8];   //!< chars used to draw border
400     int tcolor;             //!< text color
401     int bcolor;             //!< background color
402
403     //! Represents what part of frame is visible.
404     int sides_visibility[8];
405     int frame_visibility;   //!< Frame visible or not
406     coord corner_top_left;  //!< coord of top left corner
407
408     //!@{These include the border characters too
409     int height;             //!< height
410     int width;              //!< width
411     //!@}
412
413     //! Internal pmt used by operator<<
414     int state;
415
416     //! Sets the visibility of the side
417     /*!
418      /param side Specifies the side using ui::dir
419      /param visib Set the visibility of the side
420     */
421     void setside_visibility(int side, int visib);
422
423     //! Converts the ui::dir code into internally usable code
424     int convert(int);
425
426     //! Prints the frame
427     /*!
428      /param f_visib If 1, frame is printed; hidden if it's 0
429     */
430     void print(int f_visib = 1);
431
432     public:
433
434         //! Used to set the visibility mode of the frame
435         /*
436          all: ------
437               |    |
438               ------
439          nosides: -------
440
441                   -------
442         */
443         enum visibility_modes
444         {
445             all = 1,
446             nosides = 2
447         };
448
449         //! Ctor
450         /*!
451          /param corner_top_left Top left corner of frame
452          /param width Width of the frame
453          /param height Height of the frame
454         */
455         frame(coord corner_top_left = coord(1,1), int width =
```

```
456         ui::scr_width, int height = ui::scr_height - 1);
457 
458         void display(); //!< Display the frame
459         void hide();    //!< Hides the frame
460 
461         //! Sets the visibility mode of the frame
462         void setvisibility_mode(int);
463 
464         //!@{ operator<<
465         frame & operator<<(int); //!<Sets state
466 
467         //! Sets border_char according to state
468         frame & operator<<(char);
469         //!@}
470 
471         //!@{ Getter functions
472         int getheight();
473         int getwidth();
474         coord getcorner_top_left();
475 
476         //! Returns 1 if visible; 0 = not visible
477         int getframe_visibility();
478         int gettcolor();
479         int getbcolor();
480         char getborder_char(int);
481         int getside_visibility(int);
482         //!@}
483 
484         //!@{ Setter functions
485         void setheight(int);
486         void setwidth(int);
487         void settcolor(int);
488         void setbcolor(int);
489         void setcorner_top_left(coord);
490         //!@}
491 };
492 
493 //! Info related to a text box
494 /*!
495  Stores information related to a text box
496  Such as what type to convert it's data to
497  and where to store it
498 */
499 struct info_tbox
500 {
501     text_box * tbox;    //!< ptr to text_box whose info is stored
502 
503     //! Data type to convert the string stored in text box to
504     int type;
505     void * data_store;  //!< Where to store converted data
506 
507     /*!
508      A validation function that's used to validate the
509      string stored in the text box to see if it is of
510      the required type before converting it.
511      /param str The string to validate
512      /param return 1, if string is validate; 0, otherwise
513     */
514     int (*validator)(const char *str);
```

```
515
516     //! The data types the string stored in text box represents
517     /*!
518      Whenever a text box is set, the pointer to the place where
519      final data has to be stored is converted to a void* and
520      the data type is stored.
521      So, void* in different cases is:
522
523      data type      | What void* was
524      ---------------|--------------------
525      INT            | int *
526      LONG           | long *
527      UNSIGNED_LONG  | unsigned long *
528      STRING         | char *
529      CHAR           | char *
530      DOUBLE         | double *
531      FLOAT          | float *
532      PASSWORD       | char *
533     */
534     enum data_types
535     {
536         INT,
537         LONG,
538         UNSIGNED_LONG,
539         STRING,
540         CHAR,
541         DOUBLE,
542         FLOAT,
543         PASSWORD,
544         OTHER //!< Not supported at the moment
545     };
546
547     info_tbox();    //!< Ctor
548
549     //! Sets data to the data_store
550     /*!
551      Gets the string stored in the text box, validates
552      it using the validation function and then converts
553      the string to the required data type and stores it in
554      the required space
555      /return 1 on success, 0 on invalid data
556     */
557     int setdata();
558 };
559
560 /*!
561  Contains default validation functions of type
562  int f(char *)
563  that take in a string and return 1 if the string
564  is valid and 0, otherwise
565 */
566 class validation
567 {
568     validation(); //!< Object of this class is not allowed
569     public:
570
571         //!@{ Default validation functions
572         static int vint(const char *);
573         static int vlong(const char *);
```

```
574         static int vunsigned_long(const char *);
575         static int vstring(const char *);
576         static int vchar(const char *);
577         static int vdouble(const char *);
578         static int vfloat(const char *);
579         //!@}
580
581         /*!
582          Get the default validator function for the type
583          specified. If func is not NULL, returns default
584          function, else returns v
585         */
586         static validator_f getvalidator(int type,
587                                         validator_f func);
588 };
589
590 /*!
591  Represents a line with the three strings depiciting
592  left, middle and right aligned stuff respectively
593 */
594 struct line
595 {
596     //!@{ Parts of the line
597     char left[100];   //!< left aligned
598     char middle[100]; //!< centre aligned
599     char right[100];  //!< right aligned
600     //!@}
601
602     int width;  //!< width of line
603     int tcolor; //!< text color
604     int bcolor; //!< background color
605     coord corner_top_left; //!< coord of top left corner
606
607     line(); //!< Ctor
608     void display(); //!< Display the line
609     void hide();    //!< Hide the line
610     void clear();   //!< Delete the data stored
611
612     private:
613         void print(int); //!< Print the line according to arg
614 };
615
616 /*!
617  Default Back function for use in the class box.
618  Can't declare it as member function as member functions
619  are not inherently addresses and setting it as a member function
620  was causing unsolvable problems
621 */
622 int default_back_func();
623
624 //! A box that has a border and a layout
625 /*!
626  Basically incorporates all the elements into a single
627  entity that the user will interact with.
628  Basically looks like
629  ┌──────────────────┐  <── Frame
630  │ ┌──────────────┐ │
631  │ │              │ │  <──────Layout (No border)
632  │ │              │ │
```

```
633  | ---------------- <-----Padding (between layout and frame)
634   ---------------------
635 */
636 class box
637 {
638     int height;     //!< Height of the box
639     int width;      //!< Width of the box
640     int padding;    //!< Padding between frame and layout
641 
642     /*!
643      Wraps a string with specified number of characters
644      in each line
645      /param str String to wrap. Will be modified
646      /param length Number of chars in a line
647      /param return_one_line Sets string to have only one line
648      /return Number of lines after wrapping
649     */
650     int wrap(char str[], int length, int return_one_line = 0);
651 
652     //! Sets the tbox
653     /*!
654      Sets the textbox in the layout and also stores the
655      correpsonding data in a tbox that is stored in the array
656      /param data_type Type of data in text box
657      /param ptr Pointer to the data store to set in tbox
658     */
659     void set_tbox(int data_type, void *ptr);
660 
661     //!@{ Lists of interactives and text boxes
662     interactive * list_interactive[30];
663     info_tbox list_tbox[30];
664     int index_interactive; //!< Index of element to set next
665     int index_tbox; //!< Index of element to set next
666     //!@}
667 
668     //! Clicking this button exits the loop
669     button * exit_btn;
670 
671     //!@{ Toggles that help setting required info in layout
672     int center_toggle;
673     int default_toggle;
674     int right_toggle;
675     int header_toggle;
676     int footer_toggle;
677     int password_toggle;
678     //!@}
679 
680     char default_text[100]; //!< Default text to set in textbox
681 
682     /*!
683      A temporary variable that stores validator func till it
684      is stored in the required place.
685     */
686     int (*temp_validator)(const char *);
687 
688     //!@{ Header and footer
689     line header;
690     line footer;
691     //!@}
```

```
692
693     /*!
694      The function is called when the user performs a back func
695      while interacting with any interactive
696      /return 1, if loop exits on back; 0, if it does nothing
697     */
698     int (*back_func)();
699
700     protected:
701         coord pos_pointer;  //!< Pos of pointer in box
702         list_layout layout; //!< Layout in which data is stored
703         coord corner_top_left; //!< Coord of top left corner
704
705     public:
706
707         //!@{ Manipulators can be used to alter function of <<
708         static manipulator setheader;
709         static manipulator setfooter;
710         static manipulator setpassword;
711         //!@}
712
713         frame f;    //!< Border of the box
714
715         //! Ctor
716         /*!
717          Initialises all the variables of the class
718          /param corner_top_left The top left corner
719          /param width Width of box (includes border)
720          /param height Height of box (includes border)
721         */
722         box(coord corner_top_left = coord(1,1),
723             int width = ui::scr_width,
724             int height= ui::scr_height - 1);
725
726         //!@{ Getter functions
727         coord getcorner_top_left();
728         int getheight();
729         int getwidth();
730         int getpadding();
731         //!@}
732
733         //!@{ Setter functions
734         void setcorner_top_left(coord);
735         void setheight(int);
736         void setpadding(int);
737         void settcolor(int);
738         void setbcolor(int);
739         void settcolor_selected(int);
740         void setbcolor_selected(int);
741         void settcolor_input(int);
742         void setbcolor_input(int);
743         void setback_func( int(*f)(void) );
744         //!@}
745
746         //!@{ operator<< is used for adding data to the box's
747         //!   layout that will be printed
748         box & operator<<(char *);
749         box & operator<<(char);
750         box & operator<<(int);
```

```
751         box & operator<<(long);
752         box & operator<<(unsigned long);
753         box & operator<<(double);
754         box & operator<<(float);
755         box & operator<<(manipulator);
756         //!@}
757 
758         //!@{ operator>> is used for basically setting a text
759         //!   box at the place where pos_pointer is currently
760         //!   at
761         box & operator>>(char *&);
762         box & operator>>(char &);
763         box & operator>>(int &);
764         box & operator>>(long &);
765         box & operator>>(unsigned long &);
766         box & operator>>(double &);
767         box & operator>>(float &);
768         box & operator>>(manipulator);
769 
770         //! Using this before another >> will set this func
771         //! as the validator of that text box
772         box & operator>>(int (*)(const char *));
773         //!@}
774 
775         void setexit_button(char *);
776 
777         //!@{ Sets default for the next text box and
778         //!   clears it after the next text box has been
779         //!   set
780         void setdefault(char *);
781         void setdefault(char);
782         void setdefault(int);
783         void setdefault(long);
784         void setdefault(unsigned long);
785         void setdefault(double);
786         void setdefault(float);
787         //!@}
788 
789         /*!
790          Sets the box to loop, effectively enabling
791          all the text boxes and buttons. Also enables
792          scrolling
793         */
794         void loop();
795 
796         void display(); //!< Display the box
797         void hide();    //!< Hide the box
798         void clear();   //!< Delete the contents of the box
799 
800         //!@{ Functions to set header and footer
801         void setheader_tcolor(int); //!< set header color
802         void setfooter_tcolor(int); //!< set footer color
803         void clear_header(); //!< Delete contents of header
804         void clear_footer(); //!< Delete contents of footer
805         //!@}
806 };
807 
808 #endif /* UI_HPP */
```

C++ files (.cpp)

1. code/try.cpp

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

void main(){
    clrscr();
    ofstream id ("patient/max_id.dat", ios::out | ios::binary);
    long a = 0;
    id.write( (char*) &a, sizeof(a));
    getch();
}
```

2. code/iface3.cpp

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

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

```
42          menu.hide();
43          box menu2(c, ui::scr_width * 0.6, ui::scr_height - 6 );
44          menu2.clear();
45          menu2.settcolor(GREEN);
46          menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
                ;
47          menu2.settcolor(WHITE);
48          int menu2_height;
49          switch (ch)
50          {
51              case 1:
52              {
53                  menu2_height = 10;
54                  menu2.setheight(menu2_height);
55                  menu2 << "View employee data" << ui::endl;
56                  menu2.settcolor(ui::tcolor);
57                  menu2 << "Enter employee's id: ";
58                  unsigned long id;
59                  void * temp = malloc( sizeof(doctor) ); //as doctor has the
                        greatest size among employee, doctor, nurse and receptionist
                        classes
60                  if(temp == NULL)
61                  {
62                      interface::log_this("interface::employee_management() : Not
                            enough memory to allocate buffer void * temp = malloc(
                            sizeof(doctor) )");
63                      interface::error("Out of memory!! Check log");
64                      getch();
65                      break;
66                  }
67                  menu2.settcolor_input(YELLOW);
68                  menu2 >> id;
69                  menu2 << ui::endl;
70                  menu2.setexit_button("Submit");
71                  menu2.loop();
72                  menu2.hide();
73                  if(!hospital::get_employee_by_id(id, temp))
74                  {
75                      interface::error("Error while reading from file!");
76                  }
77                  else
78                  {
79                      employee *e = (employee *) temp;
80                      box menu3( menu2.getcorner_top_left(), menu2.getwidth(),
                            menu2.getheight() );
81                      menu3.setheight( menu.getheight() );
82                      menu3.settcolor(GREEN);
83                      menu3 << ui::centeralign << "Employee Management" << ui::endl
                            << ui::endl;
84                      menu3.settcolor(WHITE);
85                      menu3 << "Employee Details: " << ui::endl;
86                      menu3.settcolor(ui::tcolor);
87                      menu3 << "ID: " << e->get_id() << ui::endl;
88                      menu3 << "Name: " << e->get_name() << ui::endl;
89                      menu3 << "Age: " << e->get_age() << ui::endl;
90                      menu3 << "Sex: " << (sex)e->get_sex() << ui::endl;
91                      menu3 << "Date of Birth: " << e->get_dob() << ui::endl;
92                      menu3 << "Address: " << e->get_address() << ui::endl;
93                      menu3 << "Phone no.: " << e->get_phone() << ui::endl;
```

```
94                      menu3 << "Salary: " << e->get_salary() << ui::endl;
95                      menu3 << "Shift timings: Starts - " << e->get_shift(START) <<
                            ui::endl;
96                      menu3 << "---------------: Ends - " << e->get_shift(END) <<ui::
                            endl;
97                      switch( id_to_emp::convert( e->get_id() ) )
98                      {
99                          case INVALID:   //Test this case, menu3.hide() not
                                working properly
100                         {
101                             menu3.clear();
102                             int menu3_height = 9;
103                             menu3.setheight(menu3_height);
104                             menu3.settcolor(GREEN);
105                             menu3 << ui::centeralign << "Employee Management" <<
                                    ui::endl << ui::endl;
106                             menu3.settcolor(WHITE);
107                             menu3 << "Employee Details: " << ui::endl;
108                             menu3.settcolor(RED);
109                             menu3 << "Invalid ID!!" << id_to_emp::convert( e->
                                    get_id() );
110                             menu3.settcolor(ui::tcolor);
111                             menu3.setexit_button("Back");
112                             menu3.loop();
113                             menu3.hide();
114                             break;
115                         }
116                         case OTHERS:
117                         case RECEPTIONIST:  //there are no extra data members in
                                class receptionist
118                         {
119                             menu3.setexit_button("Back");
120                             menu3.loop();  //   menu3.clear();  int w = window.
                                    getwidth(), m = menu3.getwidth(); menu3<<w<<' '<<m
                                    ;  getch();
121                             menu3.hide();
122                             break;
123                         }
124                         case DOCTOR:
125                         {
126                             doctor *d = (doctor *)temp;
127                             menu3.hide();
128                             menu3.setcorner_top_left( coord( 1, menu3.
                                    getcorner_top_left().y ) );
129                             menu3.display();
130                             menu3.f << ( ui::top | ui::left ) << (char)204
131                                     << ( ui::bottom | ui::left ) << (char)204;
132                             menu3.f.display();
133                             box sidemenu( menu3.getcorner_top_left() + coord(
                                    menu3.getwidth() - 1, 0 ), ( ui::scr_width - menu3
                                    .getwidth() + 1 ),  menu3.getheight() );
134                             sidemenu.f << ( ui::top | ui::left ) << (char)203
135                                        << ( ui::bottom | ui::left ) << (char)202
136                                        << ( ui::top | ui::right ) << (char)185
137                                        << ( ui::bottom | ui::right ) << (char)
                                               185;
138                             sidemenu.f.display();
139                             sidemenu << "Speciality(s)" << ui::endl;
140                             for(int i = 0; i < 2 && d->get_speciality()[i] <= GEN
```

```
                                ; ++i)
141                             {
142                                 sidemenu << i + 1 << ". " << (body_parts)d->
                                        get_speciality()[i] << ui::endl;
143                             }
144                             if(!i)
145                             {
146                                 sidemenu << "None" << ui::endl;
147                             }
148                             sidemenu << "Patients currently under care:" << ui::
                                    endl;
149                             for(i = 0; d->get_patients()[i] && i < 10; ++i)
150                             {
151                                 sidemenu << i + 1 << ". " << hospital::
                                        get_patient_by_id( d->get_patients()[i] ).
                                        get_name() << ui::endl;
152                             }
153                             if(!i)
154                             {
155                                 sidemenu << "None" << ui::endl;
156                             }
157                             menu3.loop();
158                             menu3.hide();
159                             sidemenu.hide();
160                             window.f.display();
161                             break;
162                         }
163                         case NURSE:
164                         {
165                             nurse *n = (nurse *)temp;
166                             menu3.hide();
167                             menu3.setcorner_top_left( coord( 1, menu3.
                                    getcorner_top_left().y ) );
168                             menu3.display();
169                             menu3.f << ( ui::top | ui::left ) << (char)204
170                                     << ( ui::bottom | ui::left ) << (char)204;
171                             menu3.f.display();
172                             box sidemenu( menu3.getcorner_top_left() + coord(
                                    menu3.getwidth() - 1, 0 ), ( ui::scr_width - menu3
                                    .getwidth() + 1 ),  menu3.getheight() );
173                             sidemenu.f << ( ui::top | ui::left ) << (char)203
174                                        << ( ui::bottom | ui::left ) << (char)202
175                                        << ( ui::top | ui::right ) << (char)185
176                                        << ( ui::bottom | ui::right ) << (char)
                                               185;
177                             sidemenu.f.display();
178                             sidemenu << "Patients currently under care:" << ui::
                                    endl;
179                             for(int i = 0; n->get_patients()[i] && i < 5; ++i)
180                             {
181                                 sidemenu << i + 1 << ". " << hospital::
                                        get_patient_by_id( n->get_patients()[i] ).
                                        get_name() << ui::endl;
182                             }
183                             if(!i)
184                             {
185                                 sidemenu << "None" << ui::endl;
186                             }
187                             menu3.loop();
```

```
                            menu3.hide();
                            sidemenu.hide();
                            window.f.display();
                            break;
                        }
                    }
                }
                break;
            }
            case 2:
            {
                menu2 << "Add new employee" << ui::endl;
                menu2.settcolor(ui::tcolor);

                menu2.hide();
                break;
            }
            case 3:
            {
                menu2 << "Remove existing employee" << ui::endl;
                menu2.settcolor(ui::tcolor);

                menu2.hide();
                break;
            }
            case 4:
            {
                menu2 << "Edit employee data" << ui::endl;
                menu2.settcolor(ui::tcolor);

                menu2.hide();
                break;
            }
            case 5:
            {
                menu2 << "Pay salary to individual employee" << ui::endl;
                menu2.settcolor(ui::tcolor);

                menu2.hide();
                break;
            }
            case 6:
            {
                menu2 << "Pay salary to all employees" << ui::endl;
                menu2.settcolor(ui::tcolor);

                menu2.hide();
                break;
            }
            case 7:
            {
                menu2.hide();
                return;
            }
        }
    }
}
```

3. code/BASE.CPP

```
#include "base.hpp"

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

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

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

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

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

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

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

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

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

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

```
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 = 0;
94      minute = 0;
95      second = 0;
96  }
97
98  Time::Time(unsigned h, unsigned m, unsigned s)
99  {
100     if(h < 24 && m < 60 && s < 60)
101     {
102         hour = h;
103         minute = m;
104         second = s;
105     }
106     else
107     {
108         hour = 0;
109         minute = 0;
110         second = 0;
111     }
112 }
113
114 Date::Date()
115 {
116     day = 0;
```

```
117     month = 0;
118     year = 0;
119 };
120 
121 Date::Date(unsigned d, unsigned m, unsigned y)
122 {
123     if( d<=31 && m <=12)
124     {
125         day = d;
126         month = m;
127         year = y;
128     }
129     else
130     {
131         day = 0;
132         month = 0;
133         year = 0;
134     }
135 }
136 
137 Date system::get_date()
138 {
139     time_t t = time(0);
140     struct tm *now = localtime(&t);
141     Date dnow(now->tm_mday, (now->tm_mon + 1) ,(now->tm_year + 1900));
142     return dnow;
143 }
144 
145 Time system::get_time()
146 {
147     time_t t = time(0);
148     struct tm *now = localtime(&t);
149     Time tnow(now->tm_hour, now->tm_min ,now->tm_sec);
150     return tnow;
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 //  cout<<plaintext<<' '<<default_key<<endl;
159 //  cout<<strlen(plaintext)<<' '<<strlen(default_key)<<' '<<strlen(passcipher)<<
        endl;
160 //  decipher();
161 }
162 
163 userid::userid()
164 {
165     strcpy(username, "");
166     strcpy(passcipher, "");
167 }
168 
169 void userid::makecipher(str plaintext)
170 {
171     int len = strlen(plaintext);
172     int keylen = strlen(default_key);
173     for(int i = 0; i < len; ++i)
174     {
```

```
175         int plntext_i = (int)plaintext[i] + 127;
176         int key_i = (int)default_key[i % keylen] + 127;
177         passcipher[i] = (char) ( ( (plntext_i + key_i) % 256 ) - 127);
178     }
179     passcipher[i] = '\0';
180 //  cout<<passcipher<<endl<<endl;
181 }
182
183 void userid::set_key(char * plaintext)
184 {
185     randomize();
186     int len = strlen(plaintext);
187     int keylen = random(len/2 + 1) + len/2; //so that the key is not too short
188     for (int i = 0; i <=keylen; ++i)
189     {
190         default_key[i] = (char)( random(256) - 127 );
191     }
192     default_key[i] = '\0';
193 }
194
195 char * userid::decipher()
196 {
197     str decryptedpass;
198     int len = strlen(passcipher);
199     int keylen = strlen(default_key);
200     for(int i = 0; i < len; ++i)
201     {
202         int cipher_i = (int)passcipher[i] + 127;
203         int key_i = (int)default_key[i % keylen] + 127;
204         decryptedpass[i] = (char) ( ( (cipher_i - key_i + 256) % 256 ) - 127);
205     }
206     decryptedpass[i] = '\0';
207 //  cout<<endl<<decryptedpass<<endl;
208     return decryptedpass;
209 }
210
211 char * userid::get_username()
212 {
213     return username;
214 }
215
216 void userid::set_username(char * inp)
217 {
218     strcpy(username, inp);
219 }
220
221 int userid::login(char * password)
222 {
223     if(!strcmp(password, decipher()))
224         return 1;
225     else
226         return 0;
227 }
228
229 transaction::transaction(float a, Date d, Time t, char * b)
230 {
231     amount = a;
232     strcpy(reason, b);
233     _date = d;
```

```
234     _time = t;
235 }
236
237 transaction::transaction()
238 {
239     amount = 0;
240     strcpy(reason, "NA");
241     _date = Date();
242     _time = Time();
243 }
244
245 box & operator<<(box &output, sex s)
246 {
247     switch(s)
248     {
249         case MALE:
250             return output << "Male";
251         case FEMALE:
252             return output << "Female";
253         case TRANS:
254             return output << "Transsexual";
255         default:
256             return output << "Invalid";
257     }
258 }
259
260 box & operator<<(box &output, body_parts b)
261 {
262     switch(b)
263     {
264         case BRAIN:
265             return output << "Brain";
266         case HEART:
267             return output << "Heart";
268         case SKIN:
269             return output << "Skin";
270         case LUNG:
271             return output << "Lung";
272         case BONE:
273             return output << "Bone";
274         case EYE:
275             return output << "Eye";
276         case THROAT:
277             return output << "Throat";
278         case TEETH:
279             return output << "Teeth";
280         case STOMACH:
281             return output << "Stomach";
282         case BLOOD:
283             return output << "Blood";
284         case GUT:
285             return output << "Gastrointestinal tract";
286         case GEN:
287             return output << "General ailments";
288         default:
289             return output << "Invalid";
290     }
291 }
292
```

```
293 box & operator<<(box &output, Time & t)
294 {
295     return output << (unsigned long)t.hour << ':' << (unsigned long)t.minute << '
        :' << (unsigned long)t.second;
296 }
297
298 box & operator<<(box &output, Date & d)
299 {
300     return output << (unsigned long)d.day << '/' << (unsigned long)d.month << '/'
         << (unsigned long)d.year;
301 }
302
303 box & operator<<(box &output, address & a)
304 {
305     return output << a.house_no << ", " << a.street << ", "
306            << a.city << ", " << a.district << ", " << a.state;
307 }
```

4. code/HOSP.CPP

```
1  #include "hosp.hpp"
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_bal(){
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  patient hospital::get_patient_by_id(int id){
42      if(!id)
43      {
44          patient null;
45          return null;
46      }
47      str temp;
48      sprintf(temp, "patient/%d/base.dat", id);
49      ifstream patient_file ( temp , ios::in | ios::binary );
50
51      if(!patient_file){
52          // pass an error ------------------------------------------------------
53          patient b;
54          return b;
55      }
56
57      patient a;
58      patient_file.read( (char*) &a , sizeof(patient) );
59      patient_file.close();
60
61      return a;
62  }
63
64  void hospital::write_patient(patient a){
65      str temp, temp2;
66      sprintf(temp, "patient/%lu/base.dat", a.get_id());
67      sprintf(temp2, "patient/%lu", a.get_id());
68      mkdir("patient");
69      mkdir(temp2);
70      ofstream patient_file ( temp , ios::out | ios::binary );
71
72      if(patient_file){
73          patient a;
74          patient_file.write( (char*) &a , sizeof(patient) );
75      }
76      else{
77          interface::error("Patient file access failure!!");
78      }
79      if(patient_file.fail()){
80          interface::error("Patient file write failure!!");
81      }
82      patient_file.close();
83  }
84
85  void hospital::charge_patient(int pat_id, transaction trans){
86      patient temp_pat = hospital::get_patient_by_id(pat_id);
87
88      str temp;
89      sprintf(temp, "patient/%d/trans.dat", temp_pat.get_id());
90      ofstream patient_file ( temp , ios::out | ios::binary | ios::app );
91      patient_file.write( (char*) &trans , sizeof(transaction) );
92      patient_file.close();
93
94      hospital::write_patient(temp_pat);
95  }
```

```
96
97  void hospital::discharge_patient(patient temp){
98      temp.discharge();
99      temp.set_discharge_date( system::get_date() );
100     hospital::write_patient(temp);
101 }
102
103 float hospital::calc_bill(int stay){
104     return stay * ::stay_charge;
105 }
106
107 medicine hospital::get_med_by_code(int inp_code){
108     fstream meds ("hospital/medicine.dat", ios::in | ios::binary);
109
110     medicine temp;
111
112     if(inp_code < 1 || inp_code > 100){
113         temp.code = 0;
114         temp.price = 0;
115         temp.dosage = 0;
116         temp.stock = 0;
117         strcpy(temp.name, "Shell Medicine");
118
119         interface::error("Invalid medicine code!!");
120
121         return temp;
122     }
123
124     for(int i = 0; i<100; i++){
125         meds.read((char*) &temp, sizeof(medicine));
126         if(temp.code == inp_code){
127             break;
128         }
129     }
130
131     return temp;
132 }
133
134 int hospital::get_employee_by_id(unsigned long ID, void * target)
135 {
136     str temp;
137     int size_of_target;
138     switch(id_to_emp::convert(ID))
139     {
140         case INVALID:
141             interface::log_this("hospital::get_employee_by_id() : Invalid id
                    supplied to function\nFunction aborted");
142             return 0;
143         case OTHERS:
144             sprintf(temp, "employee/%lu/base.dat", ID);
145             size_of_target = sizeof(employee);
146             break;
147         case DOCTOR:
148             sprintf(temp, "employee/doctor/%lu/base.dat", ID);
149             size_of_target = sizeof(doctor);
150             break;
151         case NURSE:
152             sprintf(temp, "employee/nurse/%lu/base.dat", ID);
153             size_of_target = sizeof(nurse);
```

```
154             break;
155         case RECEPTIONIST:
156             sprintf(temp, "employee/receptionist/%lu/base.dat", ID);
157             size_of_target = sizeof(receptionist);
158             break;
159     }
160     int i = hospital::read_from( ID, (char*) target, size_of_target, temp );
161     if(!i)
162     {
163         target = NULL;
164         return 0;
165     }
166     return 1;
167 }
168 /*
169 int hospital::get_employee_by_id(unsigned long ID, doctor &target)
170 {
171     doctor null;
172     str temp;
173     sprintf(temp, "employee/doctor/%lu/base.dat", ID);
174     int i = hospital::read_from( ID, (char*) &target, sizeof(doctor), temp );
175     if(!i)
176     {
177         target = null;
178         return 0;
179     }
180     return 1;
181 }
182
183 int hospital::get_employee_by_id(unsigned long ID, nurse &target)
184 {
185     nurse null;
186     str temp;
187     sprintf(temp, "employee/nurse/%lu/base.dat", ID);
188     int i = hospital::read_from( ID, (char*) &target, sizeof(nurse), temp );
189     if(!i)
190     {
191         target = null;
192         return 0;
193     }
194     return 1;
195 }
196
197 int hospital::get_employee_by_id(unsigned long ID, receptionist &target)
198 {
199     receptionist null;
200     str temp;
201     sprintf(temp, "employee/receptionist/%lu/base.dat", ID);
202     int i = hospital::read_from( ID, (char*) &target, sizeof(receptionist), temp
        );
203     if(!i)
204     {
205         target = null;
206         return 0;
207     }
208     return 1;
209 }
210 */
211 int hospital::write_employee(void * a)
```

```
212 {
213     mkdir("employee");
214     str temp;
215     int size_of_target;
216     employee *x = (employee *) a;
217     const unsigned long ID = x->get_id();
218     switch(id_to_emp::convert(ID))
219     {
220         case INVALID:
221             interface::log_this("hospital::write_employee() : Object with ID zero
                    cannot be written to file\nFunction aborted");
222             return 0;
223         case OTHERS:
224             sprintf(temp, "employee/%lu", ID);
225             size_of_target = sizeof(employee);
226             break;
227         case DOCTOR:
228             sprintf(temp, "employee/doctor/%lu", ID);
229             size_of_target = sizeof(doctor);
230             break;
231         case NURSE:
232             sprintf(temp, "employee/nurse/%lu", ID);
233             size_of_target = sizeof(nurse);
234             break;
235         case RECEPTIONIST:
236             sprintf(temp, "employee/receptionist/%lu", ID);
237             size_of_target = sizeof(receptionist);
238             break;
239     }
240     mkdir(temp);
241     strcat(temp, "/base.dat");
242     ofstream fout ( temp , ios::out | ios::binary);
243     if(!fout)
244     {
245         interface::log_this("hospital::write_employee() : Employee data file
                could not be created or accessed\nFunction aborted");
246         return 0;
247     }
248     fout.write( (char *) a , size_of_target );
249     if(fout.fail())
250     {
251         interface::log_this("hospital::write_employee() : Error while writing to
                file (fout.fail())\nFunction aborted");
252         return 0;
253     }
254     return 1;
255 }
256 /*
257 int hospital::write_employee(doctor a)
258 {
259     mkdir("employee");
260     mkdir("employee/doctor");
261     str temp;
262     sprintf(temp, "employee/doctor/%lu", a.get_id());
263     mkdir(temp);
264     strcat(temp, "/base.dat");
265     ofstream fout ( temp , ios::out | ios::binary);
266     if(!fout)
267     {
```

```
//      cerr<<"Employee data file could not be created or accessed";
        return 0;
    }
    fout.write( (char*) &a , sizeof(doctor) );
    if(fout.fail())
    {
//      cerr<<"Error while writing to file";
        return 0;
    }
    return 1;
}

int hospital::write_employee(nurse a)
{
    mkdir("employee");
    mkdir("employee/nurse");
    str temp;
    sprintf(temp, "employee/nurse/%lu", a.get_id());
    mkdir(temp);
    strcat(temp, "/base.dat");
    ofstream fout ( temp , ios::out | ios::binary);
    if(!fout)
    {
//      cerr<<"Employee data file could not be created or accessed";
        return 0;
    }
    fout.write( (char*) &a , sizeof(nurse) );
    if(fout.fail())
    {
//      cerr<<"Error while writing to file";
        return 0;
    }
    return 1;
}

int hospital::write_employee(receptionist a)
{
    mkdir("employee");
    mkdir("employee/receptionist");
    str temp;
    sprintf(temp, "employee/receptionist/%lu", a.get_id());
    mkdir(temp);
    strcat(temp, "/base.dat");
    ofstream fout ( temp , ios::out | ios::binary);
    if(!fout)
    {
//      cerr<<"Employee data file could not be created or accessed";
        return 0;
    }
    fout.write( (char*) &a , sizeof(receptionist) );
    if(fout.fail())
    {
//      cerr<<"Error while writing to file";
        return 0;
    }
    return 1;
}
*/
int hospital::pay_salary(unsigned long id, Date d1, Time t1)
```

```
327 {
328     employee e;
329     str temp;
330 
331     if(!hospital::get_employee_by_id(id, &e))
332     {
333         interface::log_this("hospital::pay_salary() : Employee not found or error
                while reading file\nFunction aborted");
334         return 0;
335     }
336     unsigned long inp1;
337     char inp2[100] = "Salary paid to ";
338     inp1 = e.get_salary();
339     strcat(inp2, e.get_name());
340     transaction t = hospital::deduct_money(inp1, inp2, d1, t1);
341 
342     strcat(temp, "/trans.dat");
343     ofstream fout ( temp ,ios::binary | ios::app );
344     if(!fout)
345     {
346         interface::log_this("hospital::pay_salary() : Employee data file could
                not be created or accessed\nFunction aborted");
347         return 0;
348     }
349     fout.write((char *) &t, sizeof(transaction));
350     if(fout.fail())
351     {
352         interface::log_this("hospital::pay_salary() : Error while writing to file
                (fout.fail())\nFunction aborted");
353         return 0;
354     }
355     return 1;
356 }
357 
358 int hospital::pay_all_salaries()
359 {
360     Date d1;
361     d1 = system::get_date();
362     Time t1;
363     t1 = system::get_time();
364     unsigned long max_id;
365     ifstream fin;
366     fin.open("employee/max_id.dat", ios::binary);
367     if(!fin)
368     {
369         interface::log_this("hospital::pay_all_salaries() : No employees found or
                cannot access file\nFunction aborted");
370         return 0;
371     }
372     else
373     {
374         fin.read((char *) &max_id, sizeof(unsigned long));
375         if(fin.fail())
376         {
377             interface::log_this("hospital::pay_all_salaries() : Error while
                    reading file (fin.fail())\nFunction aborted");
378             return 0;
379         }
380         if(!employee::get_generate_id_status())
```

```
381         {
382             ++max_id;
383         }
384         for(unsigned long i = 1; i <= max_id; ++i)
385         {
386             int a = hospital::pay_salary(i, d1, t1);
387             if(!a)
388             {
389                 interface::log_this("hospital::pay_all_salaries() : Failed to pay
                        all salaries...\nFunction aborted");
390                 return 0;
391             }
392         }
393     }
394     return 1;
395 }
396
397 transaction* hospital::get_transaction(){
398     transaction a[10];
399
400     ifstream hosp_finances ("transactions.dat", ios::in | ios::binary);
401
402     hosp_finances.seekg( (-1) * sizeof(transaction) , hosp_finances.end );
403
404     for(int i = 0; i < 10; i++){
405         hosp_finances.read( (char *) &a[i] , sizeof(transaction) );
406         hosp_finances.seekg( hosp_finances.tellg() - ( 2 * sizeof(transaction) )
                );
407     }
408
409     return a;
410 }
411
412 int hospital::read_from(unsigned long ID, char * dest, int size, char * temp)
413 {
414     ifstream fin ( temp , ios::in | ios::binary );
415     if(!fin)
416     {
417         char errmsg[200];
418         sprintf(errmsg, "hospital::read_from() : Employee with id %lu not found\
                nFunction aborted", ID);
419         interface::log_this(errmsg);
420         return 0;
421     }
422     fin.read( dest, size );
423     if(fin.fail())
424     {
425         interface::log_this("hospital::read_from() : Error while reading from
                file (fin.fail())\nFunction aborted");
426         return 0;
427     }
428     fin.close();
429     return 1;
430 }
431
432 int hospital::count_leap_years(Date d)
433 {
434     int years = d.year;
435
```

```
436     if (d.month <= 2){
437         years--;        // checking whether to count the current year
438     }
439 
440     return (years / 4) - (years / 100) + (years / 400);
441 }
442 
443 int hospital::get_date_difference(Date dt1, Date dt2)
444 {
445 
446     long int n1 = dt1.year*365 + dt1.day;
447 
448     for (int i=0; i<dt1.month - 1; i++){
449         n1 += monthDays[i];
450     }
451     n1 += hospital::count_leap_years(dt1);
452 
453     long int n2 = dt2.year*365 + dt2.day;
454 
455     for (i=0; i<dt2.month - 1; i++){
456             n2 += monthDays[i];
457     }
458     n2 += count_leap_years(dt2);
459 
460     return (n2 - n1);
461 }
462 
463 int hospital::date_validity(const char * inp_date){
464     return hospital::date_validity(hospital::str_to_date(inp_date));
465 }
466 
467 int hospital::date_validity(Date inp_date){
468     if (
469         inp_date.month > 12 ||
470         inp_date.day > monthDays[inp_date.month - 1]
471     )
472     {
473         return 0;
474     }
475     else{
476         return 1;
477     }
478 }
479 
480 Date hospital::str_to_date(const char * inp_date){
481     int counter = 0;
482     int count = 0;
483     int input[3];
484     input[0] = input[1] = input[2] = 0;
485     while(counter < 3){
486         char ch[12];
487         ch[0] = '/';
488         for(int i = 1; i < 7; i++){
489             ch[i] = inp_date[count];
490             count++;
491             if(ch[i] == '/' || ch[i] == '\\' || ch[i] == 0 || ch[i] == '-'){
492                 if(ch[i] == 0 && count < 11){
493                     return system::get_date();
494                 }
```

```
                ch[i] = '/';
                int temp = i-1, temp2 = 0;
                while(ch[temp] != '/'){
                    input[counter] += ( pow(10, temp2) * ((int)ch[temp] - (int)'0
                        ') );
                    temp--;
                    temp2++;
                }
                counter++;
            }
        }
    }

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

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

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

double hospital::balance = 1000.0;
```

5. code/MIAN.CPP

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

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

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

    interface::init();

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

6. code/iface.cpp

```
#include <fstream.h>
#include "base.hpp"
#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;
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";
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";
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!!");
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";
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;
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 Sale";
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
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
152         case 3:
153         {
154             medicine temp;
155             temp.code = 0;
156
157             while(temp.code == 0){
158                 coord c(ui::scr_width / 3, ui::scr_height / 3);
159                 box stock_menu (c, ui::scr_width / 3, ui::scr_height / 3);
160                 stock_menu.settcolor_input(YELLOW);
161                 stock_menu << ui::centeralign << "Medicine Sale";
162                 stock_menu << "Code : ";
163                 stock_menu.setdefault(42);
164                 stock_menu >> temp.code;
165                 stock_menu << ui::endl;
166                 stock_menu.setexit_button("Submit");
167                 stock_menu.loop();
168                 stock_menu.hide();
169
170                 temp = hospital::get_med_by_code(temp.code);
171             }
172
173             coord c(ui::scr_width / 3, ui::scr_height / 3);
174             box stock_menu (c, ui::scr_width / 3, ui::scr_height / 2);
175             stock_menu.settcolor_input(YELLOW);
176             stock_menu << ui::centeralign << "Medicine Details";
177             stock_menu << "Name : " << temp.name
178                         << ui::endl << "Price : $" << temp.price
```

```
                        << ui::endl << "Dosage : " << temp.dosage << " ml"
                        << ui::endl << "Quantity in stock : " << temp.stock
                        << ui::endl;
            stock_menu.setexit_button("Okay");
            stock_menu.loop();
            stock_menu.hide();
        }
    }

}

void interface::doctor_screen()
{
    coord c(1, 4);
    box profile(c, (ui::scr_width * 3 / 5), ui::scr_height - 5);
    box menu(( c + coord((ui::scr_width * 3 / 5) - 1, 0)), (ui::scr_width * 2 /
        5) + 1, ui::scr_height - 5);
    profile.f << ( ui::top | ui::left ) << (char)204
              << ( ui::bottom | ui::left ) << (char)204
              << ( ui::top | ui::right ) << (char)203
              << ( ui::bottom | ui::right ) << (char)202;
    profile.f.display();
    menu.f << ( ui::top | ui::left ) << (char)203
           << ( ui::bottom | ui::left ) << (char)202
           << ( ui::top | ui::right ) << (char)185
           << ( ui::bottom | ui::right ) << (char)185;
    menu.f.display();
    profile.settcolor(GREEN);
    profile << ui::centeralign << "Personal Details" << ui::endl;
    profile.settcolor(ui::tcolor);
    profile << "Name: ";
    profile.setexit_button("yay");  profile.loop();menu << "yay";
    menu.hide();
    profile.hide();
    window.f.display();
}
void interface::nurse_screen()
{
    coord c(1, 4);
    box profile(c, (ui::scr_width * 3 / 5), ui::scr_height - 5);
    box menu(( c + coord((ui::scr_width * 3 / 5) - 1, 0)), (ui::scr_width * 2 /
        5) + 1, ui::scr_height - 5);
    profile.f << ( ui::top | ui::left ) << (char)204
              << ( ui::bottom | ui::left ) << (char)204
              << ( ui::top | ui::right ) << (char)203
              << ( ui::bottom | ui::right ) << (char)202;
    profile.f.display();
    menu.f << ( ui::top | ui::left ) << (char)203
           << ( ui::bottom | ui::left ) << (char)202
           << ( ui::top | ui::right ) << (char)185
           << ( ui::bottom | ui::right ) << (char)185;
    menu.f.display();
    profile.settcolor(GREEN);
    profile << ui::centeralign << "Personal Details" << ui::endl;
    profile.settcolor(ui::tcolor);
    profile << "Name: ";
    profile.setexit_button("yay");  profile.loop();menu << "yay";
    menu.hide();
    profile.hide();
```

```
236     window.f.display();
237 }
238 void interface::receptionist_screen()
239 {
240     coord c(1, 4);
241     box profile(c, (ui::scr_width * 3 / 5), ui::scr_height - 5);
242     box menu(( c + coord((ui::scr_width * 3 / 5) - 1, 0)), (ui::scr_width * 2 /
            5) + 1, ui::scr_height - 5);
243     profile.f << ( ui::top | ui::left ) << (char)204
244               << ( ui::bottom | ui::left ) << (char)204
245               << ( ui::top | ui::right ) << (char)203
246               << ( ui::bottom | ui::right ) << (char)202;
247     profile.f.display();
248     menu.f << ( ui::top | ui::left ) << (char)203
249            << ( ui::bottom | ui::left ) << (char)202
250            << ( ui::top | ui::right ) << (char)185
251            << ( ui::bottom | ui::right ) << (char)185;
252     menu.f.display();
253     profile.settcolor(GREEN);
254     profile << ui::centeralign << "Personal Details" << ui::endl;
255     profile.settcolor(ui::tcolor);
256     profile << "Name: ";
257     profile.setexit_button("yay");  profile.loop();menu << "yay";
258     menu.hide();
259     profile.hide();
260     window.f.display();
261 }
262
263 int interface::validate_menu::input(const char * ch)
264 {
265     char *endptr;
266     int a = (int) strtol(ch, &endptr, 10);
267     if(!validation::vint(ch) || a < lowest_choice || a > greatest_choice)
268     {
269         return 0;
270     }
271     else
272     {
273         return 1;
274     }
275 }
276
277 void interface::validate_menu::set_menu_limits(int a, int b)
278 {
279     lowest_choice = a;
280     greatest_choice = b;
281 }
282
283 int interface::validate_menu::lowest_choice = 0;
284 int interface::validate_menu::greatest_choice = 0;
285
286 void interface::error(char* err){
287     window.clear_footer();
288     window.setfooter_tcolor(RED);
289     window << box::setfooter << ui::centeralign
290         << err;
291 }
292
293 void interface::clear_error(){
```

```
294     window.clear_footer();
295     window.setfooter_tcolor(GREEN);
296     window << box::setfooter << ui::centeralign
297         << "Everything looks OK";
298 }
299
300 int interface::log_this(char * message)
301 {
302     Date dnow = system::get_date();
303     Time tnow = system::get_time();
304     char text[300];
305     sprintf(text, "$ [%u-%u-%u %u:%u:%u +0530]: ", dnow.day, dnow.month, dnow.
            year, tnow.hour, tnow.minute, tnow.second);
306     strcat(text, message);
307     ofstream fout;
308     fout.open("log.txt", ios::out | ios::app);
309     if(!fout)
310         return 0;
311     fout << text << endl;
312     if(fout.fail())
313         return 0;
314     fout.close();
315     return 1;
316 }
317
318 box interface::window;
```

7. code/iface2.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::patient_management();
45                      break;
46                  case 2:
47                      interface::employee_management();
48                      break;
49                  case 3:
50                      interface::stock_management();
51                      break;
52                  case 4:
53                      return;
54              }
55          }
56      }
57      else
58      {
59          switch(id_to_emp::convert(id))
60          {
61              case INVALID:
62                  interface::error("You have an invalid id generated. Create a new
                        account");
63                  break;
64              case DOCTOR:
65                  interface::doctor_screen();
66                  break;
67              case NURSE:
68                  interface::nurse_screen();
69                  break;
70              case RECEPTIONIST:
71                  interface::receptionist_screen();
72                  break;
73          }
74      }
75  }
76
77  int interface::login_screen()
78  {
79      const int login_screen_height = 9;
80      coord c(ui::scr_width / 3, ui::scr_height / 3);
81      box login_box (c, ui::scr_width / 3, login_screen_height);
82
83      str uid, pwd;
```

```
84
85      login_box.settcolor_input(YELLOW);
86      login_box << "User ID : ";
87      login_box >> uid;
88      login_box << ui::endl << "Password : ";
89      login_box >> box::setpassword >> pwd;
90      login_box << ui::endl;
91      login_box.setexit_button("Login");
92      login_box.loop();
93      login_box.hide();
94      unsigned long max_id;
95      ifstream fin;
96      fin.open("employee/max_id.dat", ios::binary);
97      if(!fin)
98          max_id = 1;
99      else
100     {
101         fin.read((char *) &max_id, sizeof(unsigned long));
102         if(fin.fail())
103         {
104             interface::error("ERROR WHILE READING FROM FILE!!! ");
105             return 0;
106         }
107     }
108     fin.close();
109     for(unsigned long id = 1; id <= max_id; ++id)
110     {
111         employee x;
112         if(!hospital::get_employee_by_id(id, &x))
113         {
114             str errmsg;
115             sprintf(errmsg, "Error in reading file of id %lu", id);
116             interface::error(errmsg);
117             return 0;
118         }
119         if(!strcmp(x.account.get_username(), uid) && x.account.login(pwd))
120         {
121             interface::clear_error();
122             return id;
123         }
124     }
125     interface::error("Invalid login details!!");
126     return 0;
127 }
128
129 int interface::menu(){
130     coord c(ui::scr_width / 3, ui::scr_height / 3);
131     box menu (c, ui::scr_width / 3, ui::scr_height / 2.2 + 1);
132
133     int ch;
134     menu << "1. Patient management"
135             << ui::endl << "2. Employee management"
136             << ui::endl << "3. Stock management"
137             << ui::endl << "4. Exit"
138             << ui::endl << ui::endl << "Choice : ";
139     menu.settcolor_input(YELLOW);
140     validate_menu::set_menu_limits(1, 4);
141     menu >> validate_menu::input >> ch;
142
```

```
143     menu << ui::endl;
144     menu.setexit_button("Submit");
145 
146     menu.loop();
147     menu.hide();
148 
149     return ch;
150 }
151 
152 void interface::patient_management(){
153     int ch = 0;
154 
155     coord c(ui::scr_width / 3, ui::scr_height / 3);
156     box menu (c, ui::scr_width / 3, ui::scr_height / 2.2);
157 
158     menu << "1. Patient admission"
159             << ui::endl << "2. Patient discharge"
160             << ui::endl << "3. Edit patient details"
161             << ui::endl << "4. Go to main menu"
162             << ui::endl << ui::endl << "Choice : ";
163     menu.setdefault(1);
164     menu.settcolor_input(YELLOW);
165     validate_menu::set_menu_limits(1,4);
166     menu >> validate_menu::input >> ch;
167 
168         menu << ui::endl;
169         menu.setexit_button("Submit");
170 
171         menu.loop();
172         menu.hide();
173 
174     switch(ch){
175         case 1:
176         {
177             coord c(ui::scr_width / 4, ui::scr_height / 4);
178             box form (c, ui::scr_width / 2, ui::scr_height / 1.25);
179             form.settcolor_input(YELLOW);
180 
181             str inp_name, inp_sex_str, inp_dob_str
182                 , inp_phone, inp_guard_name, inp_emer_contact
183                 , inp_emer_phone, inp_insur_expiry, inp_admdate_str;
184 
185             address inp_adr;
186             disease inp_dis;
187             insurance inp_insur;
188 
189             form << "Enter data for the patient :" << ui::endl
190                     << ui::endl << "Name : ";
191             form >> inp_name;
192 
193             form << ui::endl << "Sex : ";
194             form >> inp_sex_str;
195             form << ui::endl << "Key - M/F/T = Male/Female/Trans"
196                     << ui::endl << "Date of Birth : ";
197 
198             form.setdefault("25/12/1991");
199             form >> inp_dob_str;
200 
201 
```

```
202             form << ui::endl << "Address"
203                    << ui::endl << ui::endl
204                    << "\tHouse # : ";
205             form.setdefault("221B");
206             form >> inp_adr.house_no;
207
208             form << ui::endl << "\tStreet : ";
209             form.setdefault("Baker Street");
210             form >> inp_adr.street;
211
212             form << ui::endl << "\tDistrict : ";
213             form.setdefault("Idk");
214             form >> inp_adr.district;
215
216             form << ui::endl << "/tState : ";
217             form.setdefault("London(?)");
218             form >> inp_adr.state;
219
220
221             form << ui::endl << ui::endl
222                    << "Phone : ";
223             form.setdefault("1234567890");
224             form >> inp_phone;
225
226
227             form << ui::endl << "Disease"
228                    << ui::endl << ui::endl
229                    << "\tName : ";
230             form.setdefault("Melanoma");
231             form >> inp_dis.name;
232
233             form << ui::endl << "Type : ";
234             form.setdefault(0);
235             form >> inp_dis.type;
236
237             form << ui::endl << "\tType key : " << ui::endl
238                    << "\t0 - Brain\t1 - Heart" << ui::endl
239                    << "\t2 - Skin\t3 - Lung" << ui::endl
240                    << "\t4 - Bone\t5 - Eye" << ui::endl
241                    << "\t6 - Throat\t7 - Teeth" << ui::endl
242                    << "\t8 - Stomach\t9 - Blood" << ui::endl
243                    << "\t10 - General/full body condition"
244                    << ui::endl << "\tSymptoms"
245                    << ui::endl << "\tSymptom 1 : ";
246
247             form >> inp_dis.symptoms[0];
248
249             form << ui::endl << "\tSymptom 2 : ";
250             form >> inp_dis.symptoms[1];
251
252             form << ui::endl << "\tSymptom 3 : ";
253             form >> inp_dis.symptoms[2];
254
255             form << ui::endl << "\tSymptom 4 : ";
256             form >> inp_dis.symptoms[3];
257
258
259             form << ui::endl << ui::endl
260                    << "Guardian name : ";
```

```
261             form.setdefault("Dr. John Watson");
262             form >> inp_guard_name;
263 
264             form << ui::endl << "Emergency Contact : ";
265             form.setdefault("Irene Adler");
266             form >> inp_emer_contact;
267 
268             form << ui::endl << "Emer. Cont. Phone : ";
269             form.setdefault("1234567890");
270             form >> inp_emer_phone;
271 
272 
273             form << ui::endl << "Insurance"
274                    << ui::endl << ui::endl
275                    << "\tProvider : ";
276             form.setdefault("LIC");
277             form >> inp_insur.provider;
278 
279             form << ui::endl << "\tAmount ($) : ";
280             form.setdefault(30000);
281             form >> inp_insur.amount;
282 
283             form << ui::endl << "\tExpiry";
284             form.setdefault("25/12/2022");
285             form >> inp_insur_expiry;
286             inp_insur.expiry = hospital::str_to_date(inp_insur_expiry);
287 
288 
289             form << ui::endl << ui::endl
290                    << "Admission Date : ";
291             char dnow[11];
292             Date current_date = system::get_date();
293             sprintf(dnow, "%d/%d/%d", current_date.day
294                                     , current_date.month
295                                     , current_date.year);
296             form.setdefault(dnow);
297             form >> inp_admdate_str;
298 
299             form << ui::endl << ui::endl;
300             form.setexit_button("Submit");
301 
302             form.loop();
303 
304             form.hide();
305 
306             cout << "helo";
307 
308             patient temp_pat = patient(inp_name, hospital::str_to_sex(inp_sex_str
                    )
309                                        , hospital::str_to_date(inp_dob_str),
                                            inp_adr
310                                        , inp_phone, inp_dis, inp_guard_name
311                                        , inp_emer_contact, inp_emer_phone
312                                        , inp_insur, hospital::str_to_date(
                                            inp_admdate_str));
313 
314             cout << "hellllooooo";
315 
316             hospital::write_patient(temp_pat);
```

```
317
318             cout << "hellllloooo2";
319
320             break;
321         }
322
323         case 2:
324         {
325             int login_success = 0;
326
327             patient temp_patient;
328
329             while(!login_success){
330                 coord c(ui::scr_width / 3, ui::scr_height / 3);
331                 box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);
332
333                 long inp_pat_id;
334
335                 login_box << ui::endl << "Patient Discharge"
336                             << ui::endl << "Enter patient ID : ";
337                 login_box.setdefault(1);
338                 login_box >> inp_pat_id;
339
340                 login_box << ui::endl;
341                 login_box.setexit_button("Submit");
342
343                 login_box.loop();
344
345                 login_box.hide();
346
347                 temp_patient = hospital::get_patient_by_id(inp_pat_id);
348
349                 if(temp_patient.get_id() == inp_pat_id){
350                     login_success++;
351                     interface::clear_error();
352                 }
353                 else{
354                     interface::error("Invalid Patient ID!!");
355                     continue;
356                 }
357             }
358
359             coord c(ui::scr_width / 3, ui::scr_height / 3);
360             box bill (c, ui::scr_width / 3, ui::scr_height / 2);
361
362             int stay_len = hospital::get_date_difference(
363                                                     system::get_date(),
364                                                     Date(
365                                                         temp_patient.
                                                             get_admission_date
                                                             (DAY),
366                                                         temp_patient.
                                                             get_admission_date
                                                             (MONTH),
367                                                         temp_patient.
                                                             get_admission_date
                                                             (YEAR)
368                                                     )
369                                 );
```

```
370
371             bill << ui::endl << "Bill for " << temp_patient.get_name()
372                     << ui::endl << "1. Stay for "
373                     << stay_len << " days"  << ui::endl;
374
375             float total_bill;
376             bill.settcolor(GREEN);
377             bill << "$" << ( total_bill += hospital::calc_bill(stay_len) );
378
379             for(int i = 0; i < 50; i++){
380                     transaction temp_trans = temp_patient.get_transaction(i);
381
382                     if( temp_trans.amount == 0 ){
383                         break;
384                     }
385
386                     bill << i+2 << ". " << temp_trans.reason << ui::endl;
387                     bill.settcolor(GREEN);
388                     bill << "/t$" << temp_trans.amount << ui::endl;
389                     bill.settcolor(ui::tcolor);
390
391                     total_bill += temp_trans.amount;
392             }
393
394             bill.settcolor(CYAN);
395             bill << ui::endl << "Final bill : $" << total_bill;
396             bill.settcolor(ui::tcolor);
397             bill.setexit_button("Pay Bill");
398             bill.loop();
399             bill.hide();
400
401             hospital::discharge_patient(temp_patient);
402
403             break;
404         }
405
406         case 3:
407         {
408             int choice = 0, login_success = 0;
409
410             patient temp_patient;
411
412             while(!login_success){
413                 coord c(ui::scr_width / 3, ui::scr_height / 3);
414                 box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);
415
416                 long inp_pat_id;
417
418                 login_box << ui::endl << "Patient Data Alteration"
419                             << ui::endl << "Enter patient ID : ";
420                 login_box.setdefault(1);
421                 login_box >> inp_pat_id;
422
423                 login_box << ui::endl;
424                 login_box.setexit_button("Submit");
425
426                 login_box.loop();
427
428                 login_box.hide();
```

```
429
430                 temp_patient = hospital::get_patient_by_id(inp_pat_id);
431
432                 if(temp_patient.get_id() == inp_pat_id){
433                     login_success++;
434                     interface::clear_error();
435                 }
436                 else{
437                     interface::error("Invalid Patient ID!!");
438                     continue;
439                 }
440             }
441
442             while(choice < 1 || choice > 5){
443                 coord c(ui::scr_width / 3, ui::scr_height / 3);
444                 box menu (c, ui::scr_width / 3, ui::scr_height / 1.5);
445
446                 menu << "Choose item to edit:"
447                         << ui::endl << "1. Disease/condition"
448                         << ui::endl << "2. Guardian name"
449                         << ui::endl << "3. Emergency contact"
450                         << ui::endl << "4. Emergency contact no."
451                         << ui::endl << "5. Insurance information"
452                         << ui::endl << ui::endl << "Choice : ";
453                 menu.setdefault(1);
454                 menu.settcolor_input(YELLOW);
455                 menu >> choice;
456
457                 menu << ui::endl;
458                 menu.setexit_button("Submit");
459
460                 menu.loop();
461                 menu.hide();
462             }
463             switch(choice){
464                 case 1:
465                 {
466                     coord c(ui::scr_width / 3, ui::scr_height / 3);
467                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 1.5);
468
469                     edit_screen <<  "Enter disease/condition for " <<
                            temp_patient.get_name()
470                                 << ui::endl << "Disease : ";
471                     disease temp = temp_patient.get_dis();
472                     edit_screen.setdefault(temp.name);
473                     edit_screen >> temp.name;
474                     edit_screen << ui::endl << "Type : ";
475                     edit_screen.setdefault(temp.type);
476                     edit_screen >> temp.type;
477                     edit_screen << ui::endl << "Type key : " << ui::endl
478                                 << "0 - Brain\t1 - Heart" << ui::endl
479                                 << "2 - Skin\t3 - Lung" << ui::endl
480                                 << "4 - Bone\t5 - Eye" << ui::endl
481                                 << "6 - Throat\t7 - Teeth" << ui::endl
482                                 << "8 - Stomach\t9 - Blood" << ui::endl
483                                 << "10 - General/full body condition"
484                                 << ui::endl << ui::endl
485                                 << "Symptoms" << ui::endl
486                                 << "Symptom 1 : ";
```

```
487                     edit_screen.setdefault(temp.symptoms[0]);
488                     edit_screen >> temp.symptoms[0];
489                     edit_screen << ui::endl << "Symptom 2 : ";
490                     edit_screen.setdefault(temp.symptoms[1]);
491                     edit_screen >> temp.symptoms[1];
492                     edit_screen << ui::endl << "Symptom 3 : ";
493                     edit_screen.setdefault(temp.symptoms[2]);
494                     edit_screen >> temp.symptoms[2];
495                     edit_screen << ui::endl << "Symptom 4 : ";
496                     edit_screen.setdefault(temp.symptoms[3]);
497                     edit_screen >> temp.symptoms[3];
498
499                     edit_screen << ui::endl << ui::endl;
500                     edit_screen.setexit_button("Submit");
501
502                     edit_screen.loop();
503
504                     edit_screen.hide();
505
506                     temp_patient.set_dis(temp);
507                     hospital::write_patient(temp_patient);
508
509                     break;
510                 }
511
512                 case 2:
513                 {
514                     coord c(ui::scr_width / 3, ui::scr_height / 3);
515                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 3);
516
517                     edit_screen <<  "Enter name of guardian for " << temp_patient
                            .get_name()
518                                 << ui::endl << "Guardian Name : ";
519                     str temp;
520                     edit_screen.setdefault(temp_patient.get_guardian_name());
521                     edit_screen >> temp;
522
523                     edit_screen << ui::endl << ui::endl;
524                     edit_screen.setexit_button("Submit");
525
526                     edit_screen.loop();
527
528                     edit_screen.hide();
529
530                     temp_patient.set_guardian_name(temp);
531                     hospital::write_patient(temp_patient);
532
533                     break;
534                 }
535
536                 case 3:
537                 {
538                     coord c(ui::scr_width / 3, ui::scr_height / 3);
539                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 3);
540
541                     edit_screen <<  "Enter emergency contact no. for " <<
                            temp_patient.get_name()
542                                 << ui::endl << "Contact no. : ";
543                     str temp;
```

```
544                     edit_screen.setdefault(temp_patient.get_emergency_contact());
545                     edit_screen >> temp;
546
547                     edit_screen << ui::endl << ui::endl;
548                     edit_screen.setexit_button("Submit");
549
550                     edit_screen.loop();
551
552                     edit_screen.hide();
553
554                     temp_patient.set_emergency_contact(temp);
555                     hospital::write_patient(temp_patient);
556
557                     break;
558                 }
559
560                 case 4:
561                 {
562                     coord c(ui::scr_width / 3, ui::scr_height / 3);
563                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 3);
564
565                     edit_screen <<  "Enter emergency contact no. for " <<
                        temp_patient.get_name()
566                             << ui::endl << "Contact no. : ";
567                     phone temp;
568                     edit_screen.setdefault(temp_patient.get_emer_contact_no());
569                     edit_screen >> temp;
570
571                     edit_screen << ui::endl << ui::endl;
572                     edit_screen.setexit_button("Submit");
573
574                     edit_screen.loop();
575
576                     edit_screen.hide();
577
578                     temp_patient.set_emer_contact_no(temp);
579                     hospital::write_patient(temp_patient);
580
581                     break;
582                 }
583
584                 case 5:
585                 {
586                     coord c(ui::scr_width / 3, ui::scr_height / 3);
587                     box edit_screen (c, ui::scr_width / 3, ui::scr_height / 3);
588
589                     edit_screen <<  "Enter insurance information for " <<
                        temp_patient.get_name()
590                             << ui::endl << "Provider : ";
591                     insurance temp = temp_patient.get_insur_info();
592                     edit_screen.setdefault(temp.provider);
593                     edit_screen >> temp.provider;
594                     edit_screen << ui::endl << "Amount (in $) :";
595                     edit_screen.setdefault(temp.amount);
596                     edit_screen >> temp.amount;
597                     edit_screen << ui::endl << "Expiry date (MM/DD/YYYY):";
598                     char temp_date[11];
599                     edit_screen >> hospital::date_validity >> temp_date;
600
```

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

                    edit_screen.loop();

                    edit_screen.hide();

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

                    break;
                }

            }

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

8. code/EMP.CPP

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

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

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

```
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      employee temp;
80      if(hospital::get_employee_by_id(id, &temp))     //if employee's file exists
```

```
        on disk
81          hospital::write_employee(this);             //overwrite that file
82      return age;
83  }
84
85  unsigned long employee::get_salary(){
86      return salary;
87  }
88
89  void employee::set_salary(unsigned long inp)
90  {
91      salary = inp;
92  }
93
94  Time employee::get_shift(int inp){
95      switch(inp){
96          case START:
97              return shift_start;
98          case END:
99              return shift_end;
100         default:
101             return Time(0,0,0);
102     }
103 }
104
105 void employee::set_shift(int inp1, Time inp2)
106 {
107     switch (inp1)
108     {
109         case START:
110             shift_start = inp2;
111             return;
112         case END:
113             shift_end = inp2;
114             return;
115         default:
116             return;
117     }
118 }
119
120 unsigned long employee::get_id()
121 {
122     return id;
123 }
124
125 transaction * employee::get_last_10_transactions()
126 {
127     transaction t[10];
128     str temp;
129     sprintf(temp, "employee/%d/trans.dat", id);
130     ifstream fin ( temp ,ios::binary | ios::ate );
131     fin.seekg(( (-10) * sizeof(transaction) ), ios::end);
132     for(int i = 0; i < 10; ++i)
133     {
134         fin.read((char *) &t[i], sizeof(transaction));
135     }
136     return t;
137 }
138
```

```
139 int employee::get_generate_id_status()
140 {
141     return generate_id_status;
142 }
143
144 ////////////////////////////////////////
145 //// Doctor, Nurse and Receptionist class member defs
146
147 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)
148 {
149     id_to_emp i1(get_id(), DOCTOR);
150     if(!i1.status)
151     {
152         interface::error("ID not generated properly for this employee. Check log"
                );
153         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");
154     }
155     speciality[0] = inp10;
156     speciality[1] = inp11;
157
158     for(int i = 0; i < 10; i++){
159         patients[i] = 0;
160     }
161 }
162
163 doctor::doctor() : employee()
164 {
165     speciality[0] = speciality[1] = GEN + 1;    //storing an invalid value in
            speciality
166     for(int i = 0; i < 10; ++i)
167     {
168         patients[i] = 0;
169     }
170 }
171
172 int * doctor::get_speciality()
173 {
174     return speciality;
175 }
176
177 long * doctor::get_patients()
178 {
179     return patients;
180 }
181
182 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)
183 {
184     id_to_emp i1(get_id(), NURSE);
185     if(!i1.status)
186     {
187         interface::error("ID not generated properly for this employee. Check log"
                );
188         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");
```

```
189     }
190     for(int i = 0; i < 5; i++){
191         patients[i] = 0;
192     }
193 }
194
195 nurse::nurse() : employee()
196 {
197     for(int i = 0; i < 5; ++i)
198     {
199         patients[i] = 0;
200     }
201 }
202
203 long * nurse::get_patients()
204 {
205     return patients;
206 }
207
208 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)
209 {
210     id_to_emp i1(get_id(), RECEPTIONIST);
211     if(!i1.status)
212     {
213         interface::error("ID not generated properly for this employee. Check log"
                );
214         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");
215     }
216 }
217
218 receptionist::receptionist() : employee()
219 {}
220
221
222 ////////////////////////////////
223 /// Function definitions for class id_to_emp
224
225 id_to_emp::id_to_emp(unsigned long inp1, int inp2)
226 {
227     status = 0;
228     id = inp1;
229     if(!id)
230     {
231         employee_type = INVALID;
232     }
233     else
234     {
235         employee_type = inp2;
236     }
237     mkdir("employee");
238     ofstream fout;
239     fout.open("employee/id_list.dat", ios::binary | ios::ate);
240     if(!fout)
241     {
242         interface::log_this("id_to_emp::id_to_emp() : File id_list.dat couldn't
                be opened...\nFunction aborted");
```

```
243     }
244     else
245     {
246         fout.seekp(id * sizeof(id_to_emp), ios::beg);
247         fout.write((char *) this, sizeof(id_to_emp));
248         if(fout.fail())
249         {
250             interface::log_this("id_to_emp::id_to_emp() : Error while writing to
                    file id_list.dat (fout.fail())\nFunction aborted");
251         }
252         else
253         {
254             status = 1;
255         }
256     }
257 }
258
259 id_to_emp::id_to_emp()
260 {
261     id = employee_type = status = 0;
262 }
263
264 int id_to_emp::convert(unsigned long ID)
265 {
266     id_to_emp a;
267     ifstream fin;
268     fin.open("employee/id_list.dat", ios::binary);
269     if(!fin)
270     {
271         interface::log_this("id_to_emp::convert() : File id_list.dat not found!!"
                );
272         return INVALID;
273     }
274     fin.seekg( (ID * sizeof(id_to_emp)) );
275     fin.read((char *) &a, sizeof(id_to_emp));
276     if(fin.fail())
277     {
278         interface::log_this("id_to_emp::convert() : Error while reading from file
                 id_list.dat (fin.fail())");
279         return INVALID;
280     }
281     fin.close();
282     if(a.id != ID)
283     {
284         interface::log_this("id_to_emp::convert() : (For dev only)Error in the
                code... Recheck it!!");
285         return INVALID;
286     }
287     return a.employee_type;
288 }
```

9. 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 }
```

10. 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 }
```

11. 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;
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);
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)
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 }
```

12. code/UI/interact.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;
```

```
}

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

            if(key_state & 2) return SHIFT_BACKSPACE;
            else              return BACKSPACE;
        }
        case 0:     break;
        default:    return ch;
    }

    ch = getch();

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

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

    return -1;
}
```

13. code/UI/uibase.cpp

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

int init_lib_ui::counter = 0;

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

int manipulator::index = 0;
```

```
14
15 manipulator::manipulator()
16 {
17     own_index = index;
18     index++;
19 }
20
21 int manipulator::operator==(manipulator m)
22 {
23     return own_index == m.own_index;
24 }
25
26 int ui::scr_height = 0,
27     ui::scr_width = 0,
28     ui::tcolor = LIGHTGRAY,
29     ui::bcolor = BLACK;
30 manipulator ui::endl,
31             ui::centeralign,
32             ui::rightalign;
33
34 void ui::init()
35 {
36     ui::clrscr();
37
38     textcolor(ui::tcolor);
39     textbackground(ui::bcolor);
40
41     struct text_info info;
42     gettextinfo(&info);
43
44     //height and width of screen
45     scr_width = (int) info.screenwidth;
46     scr_height = (int) info.screenheight;
47 }
48
49 void ui::clrscr()
50 {
51     ::clrscr();
52 }
53
54 coord::coord(int X, int Y)
55 {
56     x = X;
57     y = Y;
58 }
59
60 coord & coord::operator+=(coord b)
61 {
62     x += b.x;
63     y += b.y;
64
65     return *this;
66 }
67
68 coord & coord::operator-=(coord b)
69 {
70     x -= b.x;
71     y -= b.y;
72
```

```
73      return *this;
74  }
75
76  coord coord::operator+(coord b)
77  {
78      coord temp = *this;
79      return temp += b ;
80  }
81
82  coord coord::operator-(coord b)
83  {
84      coord temp = *this;
85      return temp -= b;
86  }
```

14. 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)
```

```
{
    textcolor(frame::tcolor);
    textbackground(frame::bcolor);

    char visible_chars[8];
    frame_visibility = param;

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

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

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

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

    gotoxy(x, y);

    cprintf("%c", top_left);

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

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

    gotoxy(x, y + height - 1);
```

```
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 }
```

15. code/UI/box.cpp

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

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

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

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

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

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

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

    gotoxy(ctl.x + (width - strlen(middle)) / 2,
                wherey());
    if(mode == 1)
    {
```

```
58          cprintf("%s", middle);
59      }
60      else
61      {
62          for(int i = 0; i < strlen(middle); i++)
63          {
64              cprintf(" ");
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                 {
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)
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)
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
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);
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);
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         }
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 }
```

16. code/UI/infotbox.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  }
```

17. 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;
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
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 }
```

18. 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, "");
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  }
69
70  const char * list_layout_node::getstr()
71  {
72      return str;
73  }
74
75  int list_layout_node::getprint_type()
76  {
77      return print_type;
78  }
79
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;
87
88          coord ctl = getcorner_top_left();
89          if(new_y < ctl.y || new_y > ctl.y + height - 1) continue;
90
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 }
122 
123 list_layout::list_layout()
124 {
125     head = NULL,
126     current = NULL;
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 }
```

19. 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      {
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  }
```

20. 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;
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:
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 }
```

Data files

1. code/LOG.TXT
2. code/PROC.DAT
3. code/LHOSP.PRJ
4. code/LHOSP.DSK
5. code/README.md
6. code/PATIENT/MAXID.DAT
7. code/PATIENT/1/BASE.DAT
8. code/PATIENT/0/BASE.DAT
9. code/EMPLOYEE/IDLIST.DAT
10. code/EMPLOYEE/MAXID.DAT
11. code/EMPLOYEE/1/BASE.DAT