LHospital

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

Individual contributions to the project by:
Arpit Saxena 9151996

Contents

Header files

Note: The files might not be shown in their entirety. Just the contributions made by the individual are shown.

1. code/ui.hpp

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

#ifndef UI_HPP
#define UI_HPP

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

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

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

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

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

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

```
52 };
53
54 //! Class containing basic UI functions and attributes
55 class ui
56 {
57     ui();   //!< Private ctor; object of this class shouldn't be created
58     public:
59
60         //! Specifies the directions for modifying frame, etc.
61         enum dir
62         {
63             left = 1,
64             top = 2,
65             right = 4,
66             bottom = 8,
67             all = 16 //!< When all sides need to be modified
68         };
69         static int scr_height; //!< Height of screen
70         static int scr_width; //!< Width of screen
71         static void init(); //!< Sets all static variables
72         static void clrscr(); //!< Clears the contents off the screen
73         static int tcolor; //!< text color
74         static int bcolor; //!< background color
75         static manipulator endl; //!< End line and move cursor to next line
76         static manipulator centeralign; //!< Center align
77         static manipulator rightalign; //!< Right align
78
79         //! This func is called when new is unable to allocate memory
80         static void my_new_handler();
81 };
82
83 //! Represents a coordinate
84 struct coord
85 {
86     int x;  //!< x coordinate
87     int y;  //!< y coordinate
88
89     coord(int = 1,int = 1); //!< Sets the coordinate
90     coord & operator+=(coord);
91     coord & operator-=(coord);
92     coord operator+(coord);
93     coord operator-(coord);
94 };
95
96 //! Represents the node of a list representing the layout
97 /*!
98  Represents all the information of an element that will be
99  printed on the screen. Also points to the next element of the
100  screen that will be printed next to it
101 */
102 class list_layout_node
103 {
104     list_layout_node *next;     //!< Pointer to next node
105     coord pos;                  //!< Position where to print
106     int tcolor;                 //!< Text colour
107     int bcolor;                 //!< Background colour
108     char str[100];              //!< String to print
109
110     //! How to print the string; mainly for passwords
```

```
111     int print_type;
112
113     public:
114         list_layout_node();     //!< Ctor
115         ~list_layout_node();    //!< Dtor
116
117         //!@{ Setter functions
118         void setnext(list_layout_node *);
119         void setpos(coord);
120         void settcolor(int);
121         void setbcolor(int);
122         void setstr(const char *);
123         void setprint_type(int);
124         //!@}
125
126         //!@{ Getter functions
127         list_layout_node * getnext();
128         coord getpos();
129         int gettcolor();
130         int getbcolor();
131         const char * getstr();
132         int getprint_type();
133         //!@}
134
135         //! Used to distinguish will be printed i.e.
136         //! as is or hidden (as passwords)
137         enum print_types
138         {
139             DEFAULT,
140             PASSWORD
141         };
142 };
143
144 //! A node of the representation of string as a linked list
145 struct string_node
146 {
147     string_node *next;  //!< Pointer to next node
148     string_node *prev;  //!< Pointer to previous node
149     char data;          //!< Character stored in string
150
151     string_node();      //!< Ctor
152 };
153
154 //! Represents all interactive information
155 /*!
156  Basically a parent class of all the classes that
157  represent the elements of the layout the user can
158  interact with.
159  Used so that all those elements can be clubbed together
160  and the input be taken.
161 */
162 class interactive : public list_layout_node
163 {
164     interactive *prev;      //!< ptr to previous node
165     interactive *next;      //!< ptr to next node
166     int offset;             //!< offset to y position when printing
167     public:
168         interactive();      //!< Ctor
169         ~interactive();     //!< Dtor
```

```
170
171         //! Empty input function that will be overridden by children
172         /*!
173          \param offset The offset to y position
174          \return Action that was performed by the user
175         */
176         virtual int input(int offset);
177
178         //! Setter function
179         void setoffset(int);
180
181         //! Getter function
182         int getoffset();
183
184         //! Actions that are performed by user; returned from input func.
185         enum actions
186         {
187             GOTONEXT,
188             GOTOPREV,
189             CLICKED,
190             BACK //!< When shift—bckspc is pressed
191         };
192
193         //! Keys that user can press to navigate the form
194         enum keys
195         {
196             TAB,
197             ENTER,
198             BACKSPACE,
199             SHIFT_BACKSPACE,
200             SHIFT_TAB,
201             HOME,
202             END,
203             DELETE,
204             UP,
205             DOWN,
206             LEFT,
207             RIGHT
208         };
209
210         //! Gets key from user and returns code
211         /*
212          \return Keyname corresponding to enum keys
213         */
214         static int getkey();
215 };
216
217 //! Represents a text box
218 /*!
219  Inherits from interactive as a text box can be interacted
220  with. Gets data from user and stores it as a string that
221  can be further converted to the required data type
222 */
223 class text_box : public interactive
224 {
225     //! Represents if the data entered in the text box
226     //! should be displayed as is or replaced with asterisks
227     int is_password;
228
```

```
229     public:
230         text_box(); //!< Ctor
231 
232         //! Takes input and returns user action
233         /*!
234          /param offset Offset of y coordinate to print
235          /return Action performed by user
236         */
237         int input(int offset = 0);
238 
239         //! Prints string represented by a linked list
240         /*
241          Takes in the head pointer of the linked list
242          string and prints the string by iterating through
243          the list. Has no other side effects.
244          /param head ptr to head of the linked list
245         */
246         void print_str(string_node *head);
247 
248         //! Setter function
249         void setis_password(int);
250 };
251 
252 //! Represents a button that can be clicked
253 /*!
254  Inherits from interactive as a button can be interacted with.
255  A user can click the button while it's input function is
256  running which will return the user action
257 */
258 class button : public interactive
259 {
260     int tcolor_selected; //!< tcolor when selected
261     int bcolor_selected; //!< bcolor when seilected
262 
263     public:
264         button(); //!< Ctor
265 
266         //!@{ Setter functions
267         void settcolor_selected(int);
268         void setbcolor_selected(int);
269         //!@}
270 
271         //!@{ Getter functions
272         int gettcolor_selected();
273         int getbcolor_selected();
274         //!@}
275 
276         //! Input function
277         /*!
278          Effectively allows the button to be clicked
279          /param offset Offset of y coordinate to print
280          /return Action performed by the user
281         */
282         int input(int offset = 0);
283 
284         //! Prints the button
285         /*!
286          /param isselected Indicates if button is selected or not
287         */
```

```
288         void print(int isselected = 0);
289 };
290
291 //! Represents the layout of the page
292 /*!
293  Incorporates elements like simple nodes as well as other
294  interactive elements. This layout can be contained within
295  a specific height and the overflowing content can reached
296  by scrolling which is also implemented here.
297 */
298 class list_layout
299 {
300     //!@{ Pointers to implement a linked list to elements
301     list_layout_node *head; //!< ptr to head node
302     list_layout_node *current; //!< ptr to current node
303     //!@}
304
305     coord corner_top_left; //!< top left corner of container
306
307     /*!
308      Following are used as temporary placeholders till data
309      is written to the nodes
310     */
311     ///!@{
312     coord pos;
313     int tcolor;
314     int bcolor;
315     int tcolor_selected;
316     int bcolor_selected;
317     int tcolor_input;
318     int bcolor_input;
319     ///!@}
320
321     //!@{ For scrolling implementation
322     int height; //!< Height of the layout
323     int width; //!< Width of the layout
324     int lines_scrolled; //!< Lines currently scrolled
325     //!@}
326
327     //! For better verbosity at internal level
328     enum print_modes
329     {
330         DISPLAY,
331         HIDE
332     };
333
334     //! Prints the layout
335     /*!
336      Prints the layout by iterating through the internal
337      linked list maintained. Has no other side effects
338      /param print_mode How to print the data
339     */
340     void print(int print_mode = DISPLAY);
341     public:
342         list_layout(); //!< Ctor
343
344         //!@{ Set an element (node)
345         list_layout& operator<<(coord); //!< Set coord of node
346
```

```
347         //! Set data held by the node
348         list_layout& operator<<(const char *);
349         //!@}
350 
351         //! Set a text box
352         /*!
353          Sets a text box at the position indicated by pos and
354          returns a pointer to it
355          /param pos Position at which to set text box
356          /param is_pass If the text box has a password, set to 1
357          /return pointer to the text box set (casted to interactive *)
358         */
359         interactive * settext_box(coord pos, int is_pass = 0);
360 
361         //! Set a button
362         /*!
363          Sets a button at the position indicated by pos and
364          returns a pointer to it
365          /param pos Position at which to set the button
366          /param txt The text the button displays
367         */
368         interactive * setbutton(coord pos, const char *txt);
369 
370         //!@{ Setter functions
371         void settcolor(int);
372         void setbcolor(int);
373         void settcolor_selected(int);
374         void setbcolor_selected(int);
375         void settcolor_input(int);
376         void setbcolor_input(int);
377         void setcorner_top_left(coord);
378         void setheight(int);
379         void setwidth(int);
380         void setlines_scrolled(int);
381         void setpos(coord);
382         //!@}
383 
384         //!@{ Getter functions
385         int getheight();
386         int getwidth();
387         int getlines_scrolled();
388         coord getpos();
389         coord getcorner_top_left();
390         //!@}
391 
392         void display(); //!< Display the layout
393         void hide(); //!< Hide the layout
394         void clear(); //!< Deletes contents of the layout
395 };
396 
397 //! Represents a border
398 /*!
399  Basically represents a border with characters that can be
400  customised to suit the requirements.
401 */
402 class frame
403 {
404     char border_chars[8];   //!< chars used to draw border
405     int tcolor;             //!< text color
```

```
406     int bcolor;             //!< background color
407
408     //! Represents what part of frame is visible.
409     int sides_visibility[8];
410     int frame_visibility;   //!< Frame visible or not
411     coord corner_top_left;  //!< coord of top left corner
412
413     //!@{These include the border characters too
414     int height;             //!< height
415     int width;              //!< width
416     //!@}
417
418     //! Internal pmt used by operator<<
419     int state;
420
421     //! Sets the visibility of the side
422     /*!
423      /param side Specifies the side using ui::dir
424      /param visib Set the visibility of the side
425     */
426     void setside_visibility(int side, int visib);
427
428     //! Converts the ui::dir code into internally usable code
429     int convert(int);
430
431     //! Prints the frame
432     /*!
433      /param f_visib If 1, frame is printed; hidden if it's 0
434     */
435     void print(int f_visib = 1);
436
437     public:
438
439         //! Used to set the visibility mode of the frame
440         /*
441          all: ------
442               |    |
443               ------
444          nosides: -------
445
446                   -------
447         */
448         enum visibility_modes
449         {
450             all = 1,
451             nosides = 2
452         };
453
454         //! Ctor
455         /*!
456          /param corner_top_left Top left corner of frame
457          /param width Width of the frame
458          /param height Height of the frame
459         */
460         frame(coord corner_top_left = coord(1,1), int width =
461         ui::scr_width, int height = ui::scr_height - 1);
462
463         void display(); //!< Display the frame
464         void hide();    //!< Hides the frame
```

```
465
466         //! Sets the visibility mode of the frame
467         void setvisibility_mode(int);
468
469         //!@{ operator<<
470         frame & operator<<(int); //!<Sets state
471
472         //! Sets border_char according to state
473         frame & operator<<(char);
474         //!@}
475
476         //!@{ Getter functions
477         int getheight();
478         int getwidth();
479         coord getcorner_top_left();
480
481         //! Returns 1 if visible; 0 = not visible
482         int getframe_visibility();
483         int gettcolor();
484         int getbcolor();
485         char getborder_char(int);
486         int getside_visibility(int);
487         //!@}
488
489         //!@{ Setter functions
490         void setheight(int);
491         void setwidth(int);
492         void settcolor(int);
493         void setbcolor(int);
494         void setcorner_top_left(coord);
495         //!@}
496 };
497
498 //! Info related to a text box
499 /*!
500  Stores information related to a text box
501  Such as what type to convert it's data to
502  and where to store it
503 */
504 struct info_tbox
505 {
506     text_box * tbox;    //!< ptr to text_box whose info is stored
507
508     //! Data type to convert the string stored in text box to
509     int type;
510     void * data_store;  //!< Where to store converted data
511
512     /*!
513      A validation function that's used to validate the
514      string stored in the text box to see if it is of
515      the required type before converting it.
516      /param str The string to validate
517      /param return 1, if string is validate; 0, otherwise
518     */
519     int (*validator)(const char *str);
520
521     //! The data types the string stored in text box represents
522     /*!
523      Whenever a text box is set, the pointer to the place where
```

```
524      final data has to be stored is converted to a void* and
525      the data type is stored.
526      So, void* in different cases is:
527
528      data type      | What void* was
529     ________________ | ____________________
530      INT            | int *
531      LONG           | long *
532      UNSIGNED_LONG  | unsigned long *
533      STRING         | char *
534      CHAR           | char *
535      DOUBLE         | double *
536      FLOAT          | float *
537      PASSWORD       | char *
538     */
539     enum data_types
540     {
541         INT,
542         LONG,
543         UNSIGNED_LONG,
544         STRING,
545         CHAR,
546         DOUBLE,
547         FLOAT,
548         PASSWORD,
549         OTHER //!< Not supported at the moment
550     };
551
552     info_tbox();    //!< Ctor
553
554     //! Sets data to the data_store
555     /*!
556      Gets the string stored in the text box, validates
557      it using the validation function and then converts
558      the string to the required data type and stores it in
559      the required space
560      /return 1 on success, 0 on invalid data
561     */
562     int setdata();
563 };
564
565 /*!
566  Contains default validation functions of type
567  int f(char *)
568  that take in a string and return 1 if the string
569  is valid and 0, otherwise
570 */
571 class validation
572 {
573     validation(); //!< Object of this class is not allowed
574     public:
575
576         //!@{ Default validation functions
577         static int vint(const char *);
578         static int vlong(const char *);
579         static int vunsigned_long(const char *);
580         static int vstring(const char *);
581         static int vchar(const char *);
582         static int vdouble(const char *);
```

```
583         static int vfloat(const char *);
584         //!@}
585 
586         /*!
587          Get the default validator function for the type
588          specified. If func is not NULL, returns default
589          function, else returns v
590         */
591         static validator_f getvalidator(int type,
592                                         validator_f func);
593 };
594 
595 /*!
596  Represents a line with the three strings depiciting
597  left, middle and right aligned stuff respectively
598 */
599 struct line
600 {
601     //!@{ Parts of the line
602     char left[100];   //!< left aligned
603     char middle[100]; //!< centre aligned
604     char right[100];  //!< right aligned
605     //!@}
606 
607     int width;  //!< width of line
608     int tcolor; //!< text color
609     int bcolor; //!< background color
610     coord corner_top_left; //!< coord of top left corner
611 
612     line(); //!< Ctor
613     void display(); //!< Display the line
614     void hide();    //!< Hide the line
615     void clear();   //!< Delete the data stored
616 
617     private:
618         void print(int); //!< Print the line according to arg
619 };
620 
621 /*!
622  Default Back function for use in the class box.
623  Can't declare it as member function as member functions
624  are not inherently addresses and setting it as a member function
625  was causing unsolvable problems
626 */
627 int default_back_func();
628 
629 //! A box that has a border and a layout
630 /*!
631  Basically incorporates all the elements into a single
632  entity that the user will interact with.
633  Basically looks like
634  ──────────────────  <── Frame
635  │ ──────────────── │
636  │ │          <─────────Layout (No border)
637  │ │            │ │
638  │ ──────────────── <─────Padding (between layout and frame)
639  ──────────────────
640 */
641 class box
```

```
642 {
643     int height;     //!< Height of the box
644     int width;      //!< Width of the box
645     int padding;    //!< Padding between frame and layout
646 
647     /*!
648      Wraps a string with specified number of characters
649      in each line
650      /param str String to wrap. Will be modified
651      /param length Number of chars in a line
652      /param return_one_line Sets string to have only one line
653      /return Number of lines after wrapping
654     */
655     int wrap(char str[], int length, int return_one_line = 0);
656 
657     //! Sets the tbox
658     /*!
659      Sets the textbox in the layout and also stores the
660      correpsonding data in a tbox that is stored in the array
661      /param data_type Type of data in text box
662      /param ptr Pointer to the data store to set in tbox
663     */
664     void set_tbox(int data_type, void *ptr);
665 
666     //!@{ Lists of interactives and text boxes
667     interactive * list_interactive[30];
668     info_tbox list_tbox[30];
669     int index_interactive; //!< Index of element to set next
670     int index_tbox; //!< Index of element to set next
671     //!@}
672 
673     //! Clicking this button exits the loop
674     button * exit_btn;
675 
676     //!@{ Toggles that help setting required info in layout
677     int center_toggle;
678     int default_toggle;
679     int right_toggle;
680     int header_toggle;
681     int footer_toggle;
682     int password_toggle;
683     //!@}
684 
685     char default_text[100]; //!< Default text to set in textbox
686 
687     /*!
688      A temporary variable that stores validator func till it
689      is stored in the required place.
690     */
691     int (*temp_validator)(const char *);
692 
693     //!@{ Header and footer
694     line header;
695     line footer;
696     //!@}
697 
698     /*!
699      The function is called when the user performs a back func
700      while interacting with any interactive
```

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

```
760         box & operator<<(manipulator);
761         //!@}
762 
763         //!@{ operator>> is used for basically setting a text
764         //!   box at the place where pos_pointer is currently
765         //!   at
766         box & operator>>(char *&);
767         box & operator>>(char &);
768         box & operator>>(int &);
769         box & operator>>(long &);
770         box & operator>>(unsigned long &);
771         box & operator>>(double &);
772         box & operator>>(float &);
773         box & operator>>(manipulator);
774 
775         //! Using this before another >> will set this func
776         //! as the validator of that text box
777         box & operator>>(int (*)(const char *));
778         //!@}
779 
780         void setexit_button(char *);
781 
782         //!@{ Sets default for the next text box and
783         //!   clears it after the next text box has been
784         //!   set
785         void setdefault(char *);
786         void setdefault(char);
787         void setdefault(int);
788         void setdefault(long);
789         void setdefault(unsigned long);
790         void setdefault(double);
791         void setdefault(float);
792         //!@}
793 
794         /*!
795          Sets the box to loop, effectively enabling
796          all the text boxes and buttons. Also enables
797          scrolling
798         */
799         void loop();
800 
801         void display(); //!< Display the box
802         void hide();    //!< Hide the box
803         void clear();   //!< Delete the contents of the box
804 
805         //!@{ Functions to set header and footer
806         void setheader_tcolor(int); //!< set header color
807         void setfooter_tcolor(int); //!< set footer color
808         void clear_header(); //!< Delete contents of header
809         void clear_footer(); //!< Delete contents of footer
810         //!@}
811 };
812 
813 #endif /* UI_HPP */
```

C++ files (.cpp)

Note: The files might not be shown in their entirety. Just the contributions made by the individual are shown.

1. code/interact.cpp

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

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

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

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

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

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

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

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

            if(key_state & 2) return SHIFT_BACKSPACE;
```

```
52              else                return BACKSPACE;
53          }
54          case 0:     break;
55          default:    return ch;
56      }
57
58      ch = getch();
59
60      unsigned char far *key_state_byte
61      = (unsigned char far*) 0x00400017;
62      int key_state = (int) *key_state_byte;
63
64      switch(ch)
65      {
66          case 72:    return UP;
67          case 80:    return DOWN;
68          case 75:    return LEFT;
69          case 77:    return RIGHT;
70          case 15:    if (key_state & 2) return SHIFT_TAB;
71          //           ^^ Checks if shift was pressed
72          case 83:    return DELETE;
73          case 71:    return HOME;
74          case 79:    return END;
75      }
76
77      return -1;
78  }
```

2. code/uibase.cpp

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

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

```
88      return temp += b ;
89  }
90
91  coord coord::operator-(coord b)
92  {
93      coord temp = *this;
94      return temp -= b;
95  }
```

3. code/frame.cpp

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

```
48  {
49      if( visib != 0 && visib != 1)
50          return;     //No effect for invalid visibility
51
52      if(side & ui::all)
53      {
54          for(int i = 0; i < 8; i++)
55              sides_visibility[i] = visib;
56          return;
57      }
58
59      int a = frame::convert(side);
60      if(a == -1) return; //-1 indicates invalid side
61
62      sides_visibility[a] = visib;
63  }
64
65  int frame::getside_visibility(int side)
66  {
67      int a = convert(side);
68
69      if(a == -1) return -1; //Wrong side selected
70
71      return sides_visibility[a];
72  }
73
74
75  frame::frame(coord topleft, int w, int h)
76  {
77      for(int i = 0; i < 8; i++)
78      {
79          border_chars[i] = '*';
80          sides_visibility[i] = 1;
81      }
82      tcolor = ui::tcolor;
83      bcolor = ui::bcolor;
84      frame_visibility = 0;
85      height = h;
86      width = w;
87      state = 0;
88      corner_top_left = topleft;
89  }
90
91  void frame::display()
92  {
93      print(1);
94  }
95
96  void frame::hide()
97  {
98      print(0);
99  }
100
101 void frame::print(int param)
102 {
103     textcolor(frame::tcolor);
104     textbackground(frame::bcolor);
105
106     char visible_chars[8];
```

```
107     frame_visibility = param;
108
109     int x = corner_top_left.x,
110         y = corner_top_left.y;
111
112     int arr[] = {
113                 ui::top,
114                 ui::bottom,
115                 ui::left,
116                 ui::right,
117                 ui::top | ui::left,
118                 ui::top | ui::right,
119                 ui::bottom | ui::left,
120                 ui::bottom | ui::right
121             };
122
123     char &top = visible_chars[0],
124          &bottom = visible_chars[1],
125          &left = visible_chars[2],
126          &right = visible_chars[3],
127          &top_left = visible_chars[4],
128          &top_right = visible_chars[5],
129          &bottom_left = visible_chars[6],
130          &bottom_right = visible_chars[7];
131
132     for(int i = 0; i < 8; i++)
133     {
134         if(param == 1 && getside_visibility(arr[i]))
135         {
136             visible_chars[i] = getborder_char(arr[i]);
137         }
138         else
139         {
140             visible_chars[i] = ' ';
141         }
142     }
143
144     gotoxy(x, y);
145
146     cprintf("%c", top_left);
147
148     for(i = 1; i < width - 1; i++)
149     {
150         cprintf("%c", top);
151     }
152     cprintf("%c", top_right);
153
154     for(i = 1; i < height - 1; i++)
155     {
156         gotoxy(x, y + i); cprintf("%c", left);
157         gotoxy(x + width - 1, y + i); cprintf("%c", right);
158     }
159
160     gotoxy(x, y + height - 1);
161     cprintf("%c", bottom_left);
162     for(i = 1; i < width - 1; i++)
163     {
164         cprintf("%c", bottom);
165     }
```

```
166     cprintf("%c", bottom_right);
167
168     gotoxy(corner_top_left.x, corner_top_left.y);
169
170     textcolor(ui::tcolor);
171 }
172
173 void frame::setvisibility_mode(int param)
174 {
175     frame::setside_visibility(frame::all, 1);
176     if(param & nosides)
177     {
178         frame::setside_visibility(ui::left, 0);
179         frame::setside_visibility(ui::right, 0);
180     }
181     frame::display();
182 }
183
184 //Operator << is used to set border char
185 frame & frame::operator<<(int side)
186 {
187     int a = frame::convert(side);
188
189     if(a == -1) return *this; //-1 indicates error
190
191     state = a;
192
193     return *this;
194 }
195
196 frame & frame::operator<<(char border_char)
197 {
198     border_chars[frame::state] = border_char;
199     return *this;
200 }
201
202 int frame::getheight()
203 {
204     return height;
205 }
206
207 int frame::getwidth()
208 {
209     return width;
210 }
211
212 coord frame::getcorner_top_left()
213 {
214     return frame::corner_top_left;
215 }
216
217 int frame::getframe_visibility()
218 {
219     return frame_visibility;
220 }
221
222 int frame::gettcolor()
223 {
224     return tcolor;
```

```
225 }
226
227 int frame::getbcolor()
228 {
229     return bcolor;
230 }
231
232 char frame::getborder_char(int side)
233 {
234     int a = convert(side);
235
236     if(a == -1) return '\0'; //Error
237
238     return frame::border_chars[a];
239 }
240
241 void frame::setheight(int h)
242 {
243     if(h > ui::scr_height) return;
244
245     hide();
246     frame::height = h;
247     display();
248 }
249
250 void frame::setwidth(int w)
251 {
252     if(w > ui::scr_width) return;
253
254     hide();
255     frame::width = w;
256     display();
257 }
258
259 void frame::settcolor(int c)
260 {
261     tcolor = c;
262     display();
263 }
264
265 void frame::setbcolor(int b)
266 {
267     bcolor = b;
268     display();
269 }
270
271 void frame::setcorner_top_left(coord c)
272 {
273     hide();
274     frame::corner_top_left = c;
275     display();
276 }
```

4. code/box.cpp

```
1 #include "ui/ui.hpp"
2 #include "iface.hpp"
3
```

```
4  line::line()
5  {
6      strcpy(left, "");
7      strcpy(middle, "");
8      strcpy(right, "");
9      width = ui::scr_width - 2;
10     tcolor = ui::tcolor;
11     bcolor = ui::bcolor;
12     corner_top_left = coord(0,0);
13 }
14
15 void line::display()
16 {
17     print(1);
18 }
19
20 void line::hide()
21 {
22     print(0);
23 }
24
25 void line::clear()
26 {
27     hide();
28     strcpy(left, "");
29     strcpy(middle, "");
30     strcpy(right, "");
31     display();
32 }
33
34 void line::print(int mode)
35 {
36     coord curr_pos = coord(wherex(), wherey()),
37     &ctl = corner_top_left;
38     gotoxy(ctl.x, ctl.y);
39     textcolor(tcolor);
40     textbackground(bcolor);
41
42     if(mode == 1)
43     {
44         cprintf("%s", left);
45     }
46     else
47     {
48         for(int i = 0; i < strlen(left); i++)
49         {
50             cprintf(" ");
51         }
52     }
53
54     gotoxy(ctl.x + (width - strlen(middle)) / 2,
55                 wherey());
56     if(mode == 1)
57     {
58         cprintf("%s", middle);
59     }
60     else
61     {
62         for(int i = 0; i < strlen(middle); i++)
```

```
63          {
64              cprintf(" ");
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 }
```

5. code/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 }
```

6. code/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 }
```

7. code/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  }
```

8. code/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;
```

```
}
```

9. code/infotbox.cpp

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

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

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

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

```
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  }
```