The problem is to design and implement classes that simulate user interface components. In particular, the program will model windows (`CWindow`), buttons (`CButton`), input boxes (`CInput`), labels (`CLabel`), and comboboxes (`CComboBox`).

This assignment is focused on class design, where inheritance, polymorphism and abstract methods are used. If these OOP paradigms are used correctly, the implementation is short and clean. On the other hand, if the design is wrong, the implementation will be lengthy with duplicate code.

The designed classes only model the GUI. The classes do not render anything, nor they interact with the user. Indeed, the minimalistic interface only allows the instances to be assembled into a window, the components could be modified in their state and the state can be displayed in an output stream (see below). The individual components (controls) may be identified by an integer `id`. If the `id` is unique in the window, the `id` may be used to search the component (there is a method in the window interface). On the other hand, if `id` is not unique in the window, the component will work correctly, however, the searching will not work for the component.

The window must obey repositioning/resizing requests. The position and size of individual window components is a relative value, relative to the width and height of the parental window. In other words, the size and position of window components is a fractional number in the range from 0 to 1. When the position/size of the window changes, the absolute coordinates of all components must be recalculated. In our simple implementation, the absolute position is a simple product of the fractional and the window width/height. The absolute position is displayed in the output listings.

The interface of the components is similar. There is a constructor that takes either 2 or 3 parameters. The first parameter is an integer - the `id`. Then, there is the relative position of the control (the four fields in the `CRect` structure). Finally, the last argument is a string that holds the text to be contained in the component (the string argument does not apply to the combobox). Next, all components shall support deep copying and the output operator <<, the output format is shown below. The testing environment requires the constructors, copying, and the output operator for all components. There are components where further interface is required (see the listing below). Your implementation, however, may need additional further interface, you are free to extend the interface as needed.

`CButton`
: Represents a button. The testing environment only requires constructor and the output operator.

`CInput`
: Represents a text input. The last constructor parameter is the text displayed in the control. In addition to the constructor and output operator, the testing environment requires two further methods:
  - `SetValue ( x )`, which sets the string being edited to `x`,
  - `GetValue ( )`, which returns the string being edited.

`CLabel`
: Represents a static text. The last constructor parameter is the string to be displayed. The testing environment only requires constructor and the output operator.

`CComboBox`

Represents a combobox. The testing environment requires the constructor, the output operator and three additional methods:

- `Add (x)`, which adds an option (string `x`) into the list,
- `SetSelected (x)`, which selects option with index `x` (option with index 0 is selected when the combobox is initialized),
- `GetSelected ()`, which reads the index of the actually selected option.

`CWindow`

Represents a window. The interface is:

- constructor `CWindow(title, absPos )`, creates a window with label `title`. The window will be positioned on coordinate `x,y` and the size will be `w x h` (these are the members of `CRect`). Caution: the coordinates are treated as absolute coordinates.
- Method `Add(x)` adds a control `x`.
- Method `SetPosition ( newAbsPos )` modifies position/size of the window. The values are represent absolute coordinates. The method recalculates the positions/sizes of all components included in the window.
- Method `Search(id)` searches the window for a component of given `id`. If it fails (i.e. component with that id does not exist), it returns `NULL`. If there is more components with that id, the method returns a reference to either of them (e.g. to the first).
- Operator << displays the window content into the given output stream. The format is shown in the sample below.

`CRect`

is a simple helper class that implements a rectangle. The rectangle is described by its upper left corner (x,y), width (w) and height (h). The class is implemented in the testing environment, a copy of the class is included in the attached source file. Please note that the class must be kept in the conditional compile block.

Submit a source code with the implementation of classes `CWindow`, `CButton`, `CInput`, `CLabel`, and `CComboBox`. All required auxiliary declarations/functions shall be included in the source file submitted. The #include preprocessor definitions and your tests shall be placed in the conditional compile blocks (as shown in the attached file).