Unit-6 Toolkit

Solutions

6.1 Visual Toolkits

Q1. Discuss the role of visual toolkits in application programming.

Answer :

Visual Toolkits

Visual toolkits has an important role in application programming. Implementation of visual interfaces is possible with library of functions that implement building blocks like menus, buttons, dialog boxes etc. It is difficult to perform low-level programming of visual interfaces because each and every pixel on the screen has to be defined. The operating system offers as abstraction at a simple level which is far away from pixels. Even with this simple level abstraction the developer's programming is at a level which is very low.

After the creation of any application the programmer has to include additional coding, so that the application can respond to the external events given by the user. This is possible if high level abstraction is preferred. For this, X window system and Microsoft windows provide toolkits consisting of 'building bricks' to construct visual interfaces. Such building bricks are called as 'widgets' in X window system and 'controls' in Microsoft windows toolkits and widgets are an improvement in the level of programming but its implementation is complex. The developers must be proficient in C or C++ and must have the knowledge of the operating system that is currently in use so as to handle the external events well.

6.2 Fundamental Concepts of Toolkits

Q2. Tabulate the important widget classes.

Answer : **Model Paper-II, Q6(a)**

Important widget classes and their functionality is tabulated below,

Widget class	Functionality
Label	It is a frame containing one-line text
Frame	It is a rectangular container that holds widgets
Entry	It is a box used to enter a text that can be edited later
Message	It is a frame holding multi-lines text
Button	It is a label that responds to mouse clicks.
Listbox	It is a box that displays multiple-lines of text and enables the user to select any one line at any instant
Scrollbar	It is a bar that interacts with other widgets to identify what actually is displayed.
Text	It is a widget that enables editing and displaying of multiple-lines of text
Canvas	It is a container that holds images or something that has be drawn into it.
Menu	It is a widget that includes drop-down menus, pop-up menus and cascading menus.
Scale	It is used to drag a pointer either horizontally or vertically to change the integral value.

Q3. Write short notes on,

(a) Widget names

(b) Widget creation

(c) Widget properties

(d) Widget hierarchies

Answer :

(a) Widget Names

Widget name is the name of the widget that is given soon after its creation. It must be unique among all the other widgets. It must lie in the same level in the hierarchy.

(b) Widget Creation

Just by giving the widget class's name one can create an instance of the class. The widget name itself is a command.

Example: frame .fr creates an instance of a frame named fr

The names following the dot must begin with lower-case

(c) Widget Properties

Widget properties are set as soon as the widgets are created. The properties signify the appearance of the widgets like its colour, font size etc. Many widgets have common properties like size, colour, orientation and appearance. The properties can be changed accordingly. After the creation of a widget, a command with same name is defined instantly for the modification of the properties later. The instant creation of the command with same name is shown below,

```
frame.fr-text Go
...
.fr configure -text stop
```

(d) Widget Hierarchies

Hierarchy is formed from the widgets that build up an interface. It is represented in the same way how file system paths are represented with this representation a fully qualified name locates the correct location of the named widget. The hierarchy is top-level window-based i.e., main window of the application. It is created as soon as the TK application initiates by calling the TK_Main function from the C program of the application. The main function will develop the main window and initializes various possible things. Then it calls TK_Mainloop and initiates the event loop. The top-level window is an implicit frame, whose name is followed by a dot "."successive levels in the hierarchy are represented by successive dots. Consider the hierarchy below,

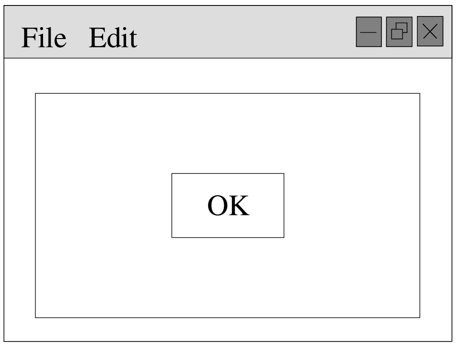


Figure: Nesting of Widgets

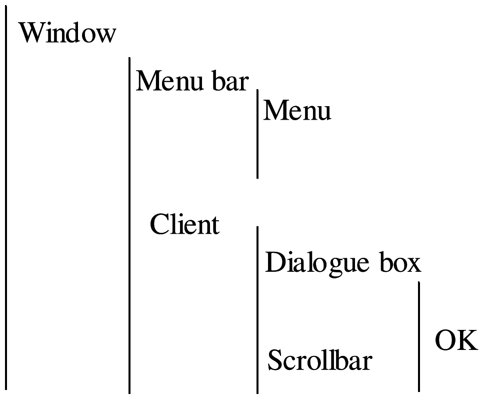


Figure: Widget Naming Hierarchy

Hence, the fully qualified name of the OK button is .clientspace.dialog.Okbutton.

Q4. List the three types of geometry managers and the commands by which they are called.

Answer :

Geometry manager is the controller of the widgets positioning. It places the widgets within a frame. One widget can be controlled by only one geometry manager. But the geometry manager has the capability to control soo many widgets in a frame. Therefore different geometry managers can control different frames. TK provides the following three kinds of geometry managers,

(a) Pack

It is a constraint-based geometry manager. The constraints are from the user's side based on the position of the widgets. For example, stacking of the widget either horizontally or vertically. After the position has been specified the geometry manager has to select an appropriate location for placement of the widget.

(b) Grid

It allows the placement of widgets on the grid with rows and columns of different size. The rows and columns are specified by the user and the size is adjusted by the geometry manager inorder to fit the widget in the grid.

(c) Place

It allows the accurate positioning of the widgets by means of absolute or relative *x* and *y* coordinates.

The above three geometry managers are called by pack, grid and place commands that accepts list of widgets, as an argument along with options. The list of widgets and the options are responsible for fine tuning the position.

6.3 Toolkits by Example

Q5. Given an example of simple button using pack and grid geometry managers also give the usage of timer events.

Answer :

In Tk, after the widget has been created, a command with the same name is defined automatically to configure the widget later. In the creation of a simple button widget, a button is created generating an interface specifying a proper label for it. The most common use of a button is to click it. So include a command option with a script that executes after the button has been clicked, after the button click the configure option changes the text on the button. Consider the following code below,

```
button .b -text “SUBMIT” \
      -command {. b configure -text “OK”}
Pack .b
```

The above code generates an interface with a label “SUBMIT”. This is shown in the following figure,

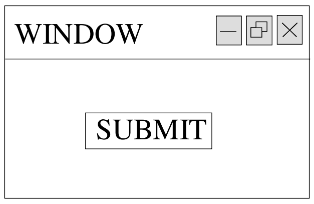


Figure: Button before Clicking

After the button has been clicked the configure option changes the label on the button form “SUBMIT” to “OK”. The script for this operation is written in the “Command” option. The following figure shows the mouse click.

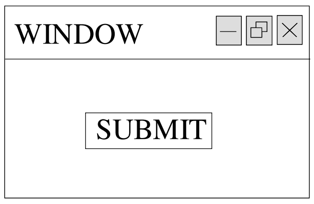


Figure: Button after Clicking

The pack geometry manager shrinks the client space vertically to fit round the button. Like wise, the grid geometry manager also produces the same output of the client space shrinking to fit the widget. The code for this as follows,

```
button .b –text “SUBMIT” \
      – command { .b configure – configure – text “OK”}
grid .b
```

Timer Events

Timer events can be generated using buttons. Consider the following example below,

```
button .b – text “SUBMIT” – Command {
        .b configure – text “OK”;
        after 3000 {bell; exit}
    }
Pack .b
```

3 seconds after the button click, a beep sound is heard and the “wish” window vanishes. The “after” command is returned soon after the delayed action event is set. The terminal bell is rung due to the “bell” command.

Q6. Given an example of entry widgets using pack and grid geometry managers

Answer :

Entry widget is a box wherein a single line text can be typed and edited. The entry command specifies a “text variable” option that specifies the text to be written in the widget. The entry commands also have the options to set the width and the appearance. There are Actions that use “bind” command that accepts widget name, the underlying action and a call back script as arguments inorder to link to the Action.

Consider the following example of a visual front-end phonebook. On entering the userid and pressing Return displays the phone number that matches with that userid. Consider the following code to set up the visual interface.

```
Label .l1    – text “Username:”
entry .el    – width 15 – relief sunken \
        – text variable euname
label .l2    – text “PhoneNo:”
entry .e2    – width 15 – relief sunken \
        – text variable ephnno
button .     – text OK – command OK
pack .l1     .e1    .l2 \
        .e2  .Ok    – side top – pady 2
bind  .e1   <Return> {set ephnno \ [search – by – name $euname] }
bind  .e2   <Return> {set uname \ [search – by – number $ephnno] }
```

The array from the file is set by the following procedure:

```
set is[open {e: \ phones.text} p}
array set phones[split [read $ in nonewline] “\n\t”]
```

The action/callback to search by name is a follows:

```
    Proc    search_by_name{e 1}{
            global phones
            if  { [info exists phones($e1) ] }
            {
                return $ phones($e1)
            }
            else
            {
                bell; return “NO INFORMATION”
```

```
                    }
                }
```

The action/callback to search by number is as follows:

```
        Proc    search_by_number{e2} {
                global  phones
                set e1 "NO INFORMATION"
                for each key[array names phones]
              {
                if {$phonebook($key) = = $e2}
              {
                set e1 $key; break
              }
        }
              if($e1 = = "NO INFORMATION"} {bell}
              return $ e1
        }
```

To stack the widgets vertically from top-to-bottom grid command needs five calls to the grid geometry manger. This is because every grid configure command begins a new row in the grid. Hence pack command in the visual interface script is replaced by the following:

```
grid configure    .l1    – pady2
grid configure    .e1    – pady2
grid configure    .l2    – pady2
grid configure    .e2    – pady2
grid configure    .Ok    – pady2
```

Q7. Give an example of listbox and explain its interaction with scrollbars.

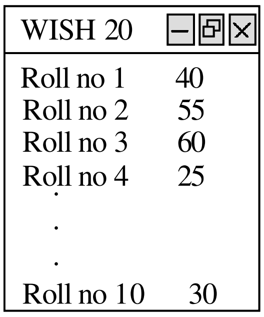
Answer :

List Box

The code to create a simple listbox is as follows,

```
List box .marks
pack .marks
Set m [open{e:\marks.txt}l]
While {[gets $m line]>=0}
{
        .marks insert end $line
}
```

The output for the above code may be expected as shown below,



```
WISH 20
Roll no 1     40
Roll no 2     55
Roll no 3     60
Roll no 4     25
   .
   .
   .
Roll no 10     30
```

Figure: List Box

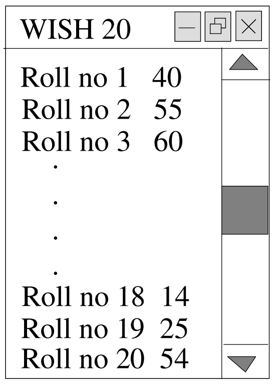
Since the default size of the listbox is 10, only the first 10 elements are visible. The while loop in the above code reads one line at a time from the file "marks". It inserts this line in the variable line. Then, this line is inserted in listbox .marks

Interaction of Listbox and Scrollbars

Scrollbar is a widget that is added to the listboxes to view its entire content. The code for Scrollbars in a listbox is shown below,

```
listbox .marks –yscrollcommand ".scroll set"
pack .marks – side left
scrollbar .scroll – command ".marks yview"
pack .scroll – side left – fill y
set m [open {e:\2ndyr\2ndsem\CN.txt}l]
while {[gets $m line] >=0}
{
        .marks insert end $line
}
```

The output for the above code may be expected as shown below,



```
WISH 20
Roll no 1   40
Roll no 2   55
Roll no 3   60
   .
   .
   .
   .
Roll no 18  14
Roll no 19  25
Roll no 20  54
```

Figure: Scrollbar in Listbox

The solid bar in the above figure represents the total size of the listbox that is to be displayed. On clicking or dragging it moves the list to the desired position. On clicking the up arrow or down arrow moves the list one position up or down the listbox.

Listbox has a – command option that specifies a partial command to be invoked when the data in the listbox change. The partial command i.e., scroll set is followed by the arguments given by the listbox. This command redraws the scrollbar. It has the following full form: widget set first last.

First and last accepts floating point values between 0 and 1. They give the relative position of the data or the element that is to be displayed if the elements in the listbox change, it calculates new values for first and last to append them to the scroll set command. This displays the scrollar again with a solid block in its exact position.

Scrollbar has a – command option that specifies a partial command to be invoked when the scrollbar is being clicked or dragged .marks yview is the partial command that changes the data in the listbox by scrolling it to some appropriate units. The fullform of this command is widget yview scroll num units. Scrolling is upwards if num value is negative otherwise it is down wards.

If gird command is used then the widgets get placed in the centre of the window to create a layout of one row and two columns use the following command,

```
grid configure .listbox .scroll
```

The size of the scrollbar is set of default unless it is specified. Using the option – sticky the widget is placed on the specified place on the screen. Along with – sticky if the options n, s, e or w is used then the widget is aligned along the edge that is specified for its cell in the grid. On using the options ns or ew the widget is stretched to the edges specified. This can be written as grid configure .listbox .scroll – sticky ns.

6.4 Events and Bindings

Q8. Explain briefly about the "bind" command.

Answer : **Model Paper-I, Q6(b)**

Bind Command

The bind command is used to register a callback script which has to be executed on the occurance of one or more events in one or more specified windows. This list of windows is called tag.

Syntax

```
bind tag sequence script
```

Here, tag defines in which widget the event has to be recognized, the sequence defines the actions of the user by which the events raise and the script is an arbitrary TCL script which is being protected by the braces from immediate evaluation. The script will be add to the existing bindings for the tag if it is preceded by a + if not the existing binding will be replaced by it. A null script { } can be specified to remove a binding.

Q9. Write short notes on,

(a) Events

(b) Keyboard events

(c) Mouse events

Answer :

(a) Events

An event is an action performed by the user. In the delegation event model, an event is an object that shows a status i.e., state changes in a source. Events are generated when the user interacts with the elements in a graphical user interface. Some of the actions that generate events are pressing a button, entering the data via keyboard, selecting an item in a list, clicking the mouse etc. These are all the events that are generated by a user by interacting with a user interface.

Events may also be generated without interacting with user interface. For example, an event may be generated when the timer expires, a counter reaches a value, a software or hardware failure occurs, or an operation is completed.

Syntax

```
bind .w <Button – 1> {-------}
bind .w <key-space> {-------}
```

(b) Keyboard Events

Whenever a key is pressed or Released, keyboard events are generated with the help of keyboard events, we can control and perform actions or get input from the keyboard.

Syntax

```
bind .b <keyPress - x) {------}
bind .b <keyRelease - x> {------}
```

(c) Mouse Events

These are several mouse events which are given as follows,

1.	Mouse_Pressed	-	Mouse was pressed
2.	Mouse_Release	-	Mouse was released
3.	Mouse_wheel	-	Mouse wheel was moved
4.	Mouse_entered	-	Mouse entered a component.
5.	Mouse_leave	-	Mouse exited from a component.

Syntax

bind .w <Button – 1> {------}

Q10. What does the event "generate" command do? Explain about virtual events.

Answer :

Event Generate Command

The event generate command allows an event to be processed as if the operating system has reported it do.

Syntax

Event generate .b <Button-1>

The piece of code enables the event handler to be invoked for a mouse click on b(button) as if the user has actually clicked the button. It does not allow sequences and allows event specification to be single event.

Virtual Events

A virtual event is an event which is associated with multiple event sequences. It fires on the occurance of one of these sequences. For example, a virtual 'paste' event can be created in a windows application which will fire on ctrl - v or shift-insert event add <<paste>> <control-key-v>/<shift-key-insert> virtual events use double angle brackets. It can used in a bind command like other events.

Bind entry <<paste>> {%w insert [selection get]}

It can be raised by the event generate command in a script.

Q11. Write about callback scripts.

Answer : **Model Paper-II, Q6(b)**

Callback Scripts

The callback is a Tcl script, which is protected by the braces from being evaluated. It is nothing but a call of a procedure which defined some where else. The special variables which represent the attribute information which are related to an event can be accessed by this script. Hence, %b, %x and %y contain the button number and the mouse co-ordinates respectively for the mouse events at the time of the event. The %w contains the widget name for all the events. For example, the commands to turn the button red when the mouse enters and restore the default color when it leaves would be,

bind Button <enter> {%w configure\– background red}

bind Button <LEAVE> {%w configure\– background grey}

We can add the arguments inorder to provide the information to be returned in the 'percent' variables if at all an event is raised in the script with event generate.

event generate . f <Button – 1> – x 10 – y 20

Here, .f is frame which generates the mouseclick event at (10, 20) coordinates. These values will be substituted before the callback script gets evaluated. So, they can be used as procedure arguments.

bind Button <Button – 1> {btnl_click %w}

6.5 Perl Toolkit

Q12. What is perl-toolkits. Explain how it is different from Tcl/Tk.

Answer : **Model Paper-I, Q6(b)**

Perl-Tk

Perl-Tk provides the Tk functionality in perl environment allowing it to add GUI to perl script. It is designed in such a way that a novice user of it who is familiar with Tcl/Tk can quickly adapt.

Difference Between Perl-Tk and Tcl/Tk

	Perl-Tk		Tcl-Tk
1.	Perl-Tk provides the Tk functionality in perl environment allowing to add GUI to perl script.	1.	Tcl-Tk is a command language which is written in Tcl and built on-the-fly commands and procedures.
2.	It implements the widgets as objects by using a group of methods.	2.	It allows the user to create and manipulate the widgets which are a group of visual items.
3.	It uses the event and the call back script as arguments in order to invoke the widget's bind method.	3.	It uses the name of the widget, the event and the call back script as arguments along with the bind command in order to associate an event with a callback.
4.	Example: button .b -text "press Me". \ – command {exit} Pack .b	4.	Example: $mw = new Mainwindow; $b = $mw → Button (-text ⇒ "Press Me", - command ⇒ sub {exit}) → pack; Main Loop;

OBJECTIVE TYPE

I. Fill in the Blanks

1. __________ widget is a container for drawings and images.
2. __________ is a frame containing one-line text.
3. __________ geometry manager enables precise positioning of the widgets.
4. __________ is a frame that takes it's own window.
5. The full form of yview command option is __________.
6. __________ is the partial command used to redraw the scrollbar.
7. __________ is a Tcl scripted that is secured by braces from being evaluated.
8. For mouse events %x and %y represents the __________ during the time of events.
9. The __________ command processes the event as a report by the operating system.
10. To identify a button, the button events are qualified by a __________.

II. Multiple Choice

1. __________ widget contains one-line text. [ ]

   (a) Frame (b) Label
   (c) Entry (d) Text

2. __________ widget enables display and editing of multiple lines. [ ]

   (a) Frame (b) Label
   (c) Entry (d) Text

3. A simple button widget has __________ attributes. [ ]

   (a) 17 (b) 27
   (c) 37 (d) None

4. __________ geometry manager is constraint -based. [ ]

   (a) Pack (b) Unpack
   (c) Grid (d) Place

5. __________ is a handy editor for short text files. [ ]

   (a) Toolbar (b) Notepad
   (c) Window (d) Menu

6. List of windows is called __________. [ ]

   (a) Group (b) Tag
   (c) Menu (d) Widgets

7. __________ is linked with multiple event sequences and gets fired on the occurrence of any event. [ ]

   (a) Keyboard events (b) Mouse events

   (c) Both (a) and (b) (d) Virtual events

8. Tk programming style is __________. [ ]

   (a) Object – oriented (b) Platform – oriented

   (c) Action – oriented (d) Tool-kit – oriented

9. In mouse events __________ contains the button number. [ ]

   (a) %a (b) %b

   (c) %x (d) %y

10. __________ command is used to stretch the button to uniform width. [ ]

   (a) – cascade x (b) – fill x

   (c) – bind x (d) – tag x

KEY

I. Fill in the Blanks

1. Canvas
2. Label
3. Place
4. Top-level window
5. Widget yview scroll num units
6. Scroll set
7. Callback
8. Mouse coordinates
9. Event generate
10. Button number

II. Multiple Choice

1. (a)
2. (d)
3. (c)
4. (a)
5. (b)
6. (b)
7. (d)
8. (a)
9. (b)
10. (b)