Mikhail E. Abramyan

CS212. PARADIGMS AND TECHNOLOGIES OF PROGRAMMING

MODULE 2
USER INTERFACE DEVELOPMENT

Study Assignments

1. Study assignments

1.1. General requirements

If the form does not contain controls for which it makes sense to resize, then it cannot be resized or maximized.

When placing controls in a form, make optimal use of the form space. For forms of variable size, you should actively use the Anchor or Dock properties.

All actions provided in the project must be associated with keyboard shortcuts. Information about keyboard shortcuts should be available in the program window through underlined characters in the titles of controls, additional text in parentheses (for example, **Run (F5)**), default buttons. Changing the focus with the Tab key should occur in a natural order: left to right and top to bottom.

Programs should not contain handlers of the same type; instead, you must connect one handler to multiple controls. Actions applied to multiple controls must be performed in a loop using the Controls property of the form.

The default names of controls should not be changed, except for the names of menu items and shortcut buttons.

1.2. CONSOLE project: console applications, file and directory processing

General guidelines. All input data for the program must be passed using command line arguments. To test a program with different arguments, use the project settings specified in the **Debug** group and prepare a special directory structure on disk.

The program should provide formatted output (for instance, using interpolated strings). LINQ queries can be useful in a number of situations.

The program must correctly handle directories with relative paths (that is, with paths that do not start with a drive letter and a backslash). For example, specifying the **dir1** directory means that the **dir1** directory is a subdirectory of the current directory. Special directory names "." (one dot) and ".." (two dots) do not need to be processed.

At the beginning of its work, the program should print to the console an example of its call with the command line arguments (if some arguments are optional, they are enclosed in square brackets). If there is an invalid argument (for example, the name of a directory that does not exist), the program should display an error message. After the completion of the data output, it is necessary that the form does not close immediately and allows the user to view the results obtained.

1. At the end of program work, a list of files from the current directory (or the directory specified as a command line argument) is displayed in the console. Files are sorted by size. Information about files should be displayed in three columns containing the file name, size (in bytes) and creation date. You do not need to process subdirectories.

2. At the end of program work, a list of subdirectories of the current directory (or the directory specified as a command line argument) is displayed in the console. Subdirectories of all levels must be listed in the order of their nesting, and directories must be indented with four spaces for each level. At each level, the directories must be sorted alphabetically by name.

3. At the end of program work, the console displays a list of files from the current or user-specified directory with names matching the user-specified mask (for example, ***.jpg**). The directory and mask are specified by the user on the command line (the first is the mask, the second is the directory). If the second argument is missing, the current directory is processed. If all arguments are missing, an error message is displayed. Information about files should be displayed in three columns containing the file name, its size (in bytes) and the date of creation (files are sorted by name in alphabetical order). You do not need to process subdirectories.

4. At the end of program work, the console displays a list of files matching the specified mask in all-level subdirectories of the current directory (or the directory specified as the first command line argument). The mask is specified as the second argument; if it is not specified, then it is considered equal to ***.***. For each subdirectory, you should output its full name and then a list of the names of the files that satisfy the mask and are located in this subdirectory, together with their size in bytes (each name is displayed on a new line with an indention of 4 spaces, the files are sorted in alphabetical order of names). Subdirectories can be listed in any order.

5. At the end of program work, the console displays summary information about files matching the specified mask in all-level subdirectories of the current directory (or the directory specified as the first command line argument). The mask is specified as the second argument; if it is not specified, then it is considered equal to ***.***. For each subdirectory, first its name is displayed, then (in the same line) the number of files matching the mask and their total size in bytes. Subdirectories of all levels must be listed in the order of their nesting, directories must be indented with four spaces for each level. Within each level, directories can be listed in any order.

6. At the end of program work, the console displays a list of files from the user-specified directory with names that match the user-specified creation date interval (subdirectories are not processed; the start date and end date may coin-

cide). The directory, the start date, and the end date are specified by the user on the command line as three arguments. If there are less than three command line arguments, an error message is displayed. Information about files should be displayed in three columns containing the file name, its size (in bytes), and the creation date. Files must be sorted in ascending order of creation date. In an example of program call (this example should be print to the console at the beginning of program work), the required date format should be displayed.

7. Create a program that compares the contents of files with the same name (case insensitive) in the two directories specified by the user as two command line arguments. If there are less than two command line arguments, an error message is displayed. Two lists of files are displayed on the screen: (1) a list of files with the same name and the same contents, (2) a list of files with the same name and different contents. In each list, files must be sorted alphabetically, with each name displayed on a new line.

8. Create a program that compares the contents of two directories specified by the user as two command line arguments. If there are less than two command line arguments, an error message is displayed. Three lists of files must be displayed on the screen: (1) a list of files present in the first directory and absent in the second, (2) a list of files present in the second directory and absent in the first, (3) a list of files contained in both directories. File contents is not required to analyze. In each list, files must be sorted alphabetically, with each name displayed on a new line. Filename comparisons are not case sensitive. You do not need to process subdirectories.

9. Create a program that copies the structure of the nested directories. In the created directory structure, each subdirectory name must have a prefix specified by the user in the first command line argument. The source directory (containing the directory structure to be copied) and the destination directory (in which the copy of this structure will be created) must be specified as the second and third command line argument, respectively. If there are less than three command line arguments, an error message is displayed. The program must display the number of created directories. You do not need to copy files.

10. Create a program that calculates two values: the number and total size of all files located in the specified directory and its all-level subdirectories and satisfying the mask specified by the user. The top-level directory and mask are specified by the user as the first and second command line arguments, respectively. If no mask is specified, the ***.*** mask is used. If the directory is also not specified, then the current directory is processed. Several masks can be specified in the command line arguments; in this case, each mask is processed separately, and the program displays more than two values (each two values must be displayed on a new line).

11. Create a program that searches for a file by mask in the structure of nested subdirectories, excluding some subdirectories from consideration. The initial directory name, file mask, and subdirectory names to be excluded are specified by the user on the command line (there can be multiple names of excluded subdirectories). Files of the initial directory are always processed. For all found files, their full name, size (in bytes), and creation date are displayed (files can be displayed in any order). If names of excluded directories are not specified, then all subdirectories are processed; if, in addition, no file mask is specified, then all files are displayed. If all command line arguments are missing, the current directory is processed.

12. Create a program that finds the characteristics of the structure of nested directories: the total number of subdirectories, the maximum nesting depth, the maximum number of files in one directory, the total size of all files in the initial directory and all subdirectories. The initial directory is specified as a command line argument; if the initial directory is not specified, then the current directory is processed.

13. Create a program to rename all files with a user-specified name in a given directory and all its subdirectories. The user specifies the old and new file names (without * and ? wildcards) and the top-level directory as command line arguments. If there are less than three command line arguments, an error message is displayed. Searching for files to rename should not be case sensitive. The program displays the total number of renamed files.

1.3. DIALOGS project: form interaction

General guidelines. For dialog boxes (that is, modal forms), it is necessary to implement the standard actions for the Enter and Esc keys (the Enter key is always associated with the **OK** button or its analog, the Esc key is always associated with the **Cancel** key or its analog). When you reopen the dialog box, it must either retain the previous information or display new information if such is the condition of a study assignment. In any case, it is necessary to ensure that the first control of the dialog box is activated. The dialog box should not contain minimize and maximize buttons. The main form must contain an available minimize button; maximization should only be available for the resizable main form. By default (if a study assignment does not require the dialog box to be resizable), the dialog box should be fixed in size.

1. The fixed-size main form contains a text box, the **Password** label, and the **Open a protected form** button. The initial password is **qwerty**. If the password is input correctly, then, when the button (or the Enter key) is pressed, a modal form with the **Protected form** title appears containing two text boxes with the **New password** label (at the beginning, these text boxes contain the initial password) and two modal buttons: **OK** and **Cancel**. The **OK** button is avail-

able if both text boxes in the modal form contain the same non-empty text. When you close the modal form with the **OK** button or the Enter key, this text becomes the new password. When you close the modal form with the **Cancel** button or the Esc key, the password is not changed. When input a password, the symbols "*" should be displayed instead of the typed characters; to do this, use the PasswordChar property of the TextBox control. You do not need to save your password when you close the application.

2. The fixed-size main form contains the **Change scaling** button (the button is located at the upper left corner of the form). When the button is pressed, a modal form with the **Scaling** title appears containing a text box with the **New scale (%)** label and three buttons: **OK**, **Apply**, and **Cancel**. The **OK** and **Apply** buttons are available if the text box contains an integer in the range from **10** to **300**. The initial value of this input box is **100**; only digits can be input in this box. When you press the **OK** or **Apply** buttons, the main form and the button it contains change their sizes in accordance with the new scale factor (for example, if the factor is **200**, then the sizes are doubled); in case of pressing the **OK** button, the modal form will close. The scaling is always relative to the initial size of the form. When you click the **Cancel** button, the modal form closes without performing any additional action. When you reopen the modal form, the text box should display the current scale factor value.

3. The resizable main form contains a multi-line text box. When you try to close the form, a modal form appears with buttons: **Save**, **Not save**, **Cancel**. When you click the **Save** button, the entered text will be saved in the **text.txt** file. Pressing the **Cancel** button cancels the closing of the main form. When you reopen the form, if there is a **text.txt** file, its text is loaded into the text box.

4. The resizable main form contains the **Show** button. When this button is clicked, a modal form appears with two text boxes containing the current values of the coordinates of the upper left corner of the **Show** button (the **x** coordinate is displayed in the first text box, the **y** coordinate in the second) and the buttons **OK**, **Apply**, and **Cancel** . When you click the **OK** or **Apply** buttons, new coordinates are set for the **Show** button; in case of pressing the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action. If the text boxes contain invalid values, the **OK** and **Apply** buttons should be unavailable (the value is considered valid if it can be converted to a non-negative number and, after moving to the specified position, the **Show** button will be at least partially visible on the screen).

5. The fixed-size main form contains the **Twin** button. When you click the button, a modal form of the same size appears with the **OK** and **Cancel** buttons. The modal form can be resized. The **OK** button (and the Enter key) closes the

modal form and sets the size of the modal form for the main form, the **Cancel** button (and the Esc key) also closes the modal form, but does not change the size of the main form.

6. The resizable main form contains the **Add Label** button. When you click the button, a modal form appears with two text boxes for the coordinates of a new label, a text box for the text of this label, and buttons **OK**, **Apply**, and **Cancel**. The **OK** and **Apply** buttons are available if the coordinates of a new label are non-negative numbers corresponding to some point in the main form and the label text is a non-empty string. When you click the **OK** or **Apply** buttons, a new label is created in the main form in the position specified by the user; in case of clicking the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action. If the modal form was closed by pressing the **OK** button, then, when it is reopened, the coordinates for the new label should increase by **20**.

7. The fixed-size main form contains the **Toss a coin** button and labels displaying statistics with the text **Total number of tosses = 0** and **Percentage of heads = 0**. When you click the **Toss a coin** button, a modal form appears with a text box for input the number of tosses and the **OK**, **Apply**, and **Cancel** buttons. The **OK** and **Apply** buttons are available if the text box contains an integer in the range from **1** to **10000**. The initial value of this input box is **10**; only digits can be input in this box. When you press the **OK** or **Apply** buttons, the required number of coin tosses is simulated and statistics labels are updated in the main form; in case of pressing the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action. Use the Random class to simulate a coin toss.

8. The fixed-size main form contains three labels. When you click on one of the labels, a modal form appears with a text box and buttons **OK**, **Apply**, and **Cancel**. The text box contains the text of the label being clicked; this text can be changed. When you click **OK** or **Apply**, the text of the corresponding label in the main form is updated; in case of pressing the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action.

9. The fixed-size main form contains three text boxes and a label (not a button!) with the text **Change edit mode**. At the launching of the program, the first text box has focus, and all text boxes are editable. When you click on the label, a modal form appears with the label **Are you sure?** and buttons **Yes** and **No**. Clicking the **Yes** button changes the edit mode for the text box that has focus (editable mode is switched to read-only mode and vice versa). Pressing the **No** button does not change the edit mode. In any case, the modal form will close.

You should implement the modal form yourself without using the MessageBox class.

10. The fixed-size main form contains the **Change Background** button. When you click this button, a modal form appears with three text boxes for input the intensities of the red, green, and blue color components. In addition, the modal form contains buttons **OK**, **Apply**, and **Cancel**. When you click the **OK** or **Apply** buttons, the background color of the main form changes; in case of clicking the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action. If at least one of the text boxes contains text other than a number from the range 0–255, then the **OK** and **Apply** buttons should be unavailable. Use the Color structure and its FromRGB method.

11. The resizable main form contains the **Change alignment** button. When you click on this button, a modal form appears with two drop-down lists labeled **Horizontal alignment** and **Vertical alignment**. Each list contains three options: **Align Left**, **Align Center**, **Align Right** for the **Horizontal alignment** list and **Align Top**, **Align Middle**, **Align Bottom** for the **Vertical alignment** list. In addition, the modal form contains buttons **OK**, **Apply**, and **Cancel**. When you click the **OK** or **Apply** buttons, the main form changes its position on the screen in accordance with the values of the drop-down lists; in case of clicking the **OK** button, the modal form will close. When you click the **Cancel** button, the modal form closes without performing any additional action. When aligning, take into account the current size of the main form. Use the PrimaryScreen.WorkingArea property of the Screen class. Working with drop-down lists is described in the LISTBOXES project (Section 19.1).

1.4. SYNC project: control synchronization

General guidelines. In all projects, it is required to configure all controls of the same type by specifying *one* event handler for each group of controls of the same type. If statements (such as if (n == 1), if (n == 2), if (n == 3), ...) or switch statements should not be used in handlers. It is allowed to use the Tag properties of the controls, as well as the Controls property of the form, which allows you to refer to a control by its name. See the CALC project (Section 6.1) for information on sharing event handlers.

Working with *text boxes* is described in the CALC project (Section 6.5) and TEXTBOXES project (Section 8.1). Working with *checkboxes* is described in the CHECKBOXES project (Chapter 20). Working with *radio buttons* is described in the TEXTBOXES project (Section 8.2). Working with *toolbar* and *shortcut buttons* is described in the TEXTEDIT4 project (Chapter 15). Working with *track bars* is described in the COLORS project (Section 18.1). Working with NumericUpDown controls is described in the TEXTEDIT6 project (Sec-

tion 17.3). Working with *progress bars* is described in the TRIGFUNC project (Section 23.4).

1. The main form contains six text boxes with the text **1** – **6** and the **Show** button. When the **Show** button is clicked, a second (non-modal) form appears containing six checkboxes (the CheckBox controls). The checkboxes are labeled **1** – **6**; initially none of them is checked. When you check any checkbox, the text of the corresponding text box is highlighted in bold; when you uncheck the checkbox, the bold highlighting of the corresponding text box is canceled. When changing the text of any text box, the label of the corresponding checkbox should change accordingly.

2. The main form contains six text boxes with the text **1** – **6** and the **Show** button; the text of the first text box must be in bold. When the **Show** button is clicked, a second (non-modal) form appears, containing six radio buttons (the RadioButton controls). Radio buttons are labeled **1** – **6**; the radio button corresponding to the bold text box must be selected. When you select another radio button, the bold highlighting is transferred to the text of the corresponding text box. When changing the text of any text box, the label of the corresponding radio button should change accordingly.

3. The main form contains the **Show** button and the toolbar (the ToolStrip control) with 6 shortcut buttons (the ToolStripButton controls). The shortcut buttons on the toolbar have titles **1** – **6**; initially, none of the shortcut buttons is in the pressed state. When the **Show** button is clicked, a second (non-modal) form appears containing 6 checkboxes (the CheckBox controls). The checkboxes are labeled **1** – **6**. When checking/unchecking any checkbox, the corresponding shortcut button on the toolbar is automatically pressed/released; when the shortcut button is pressed/released on the toolbar, the corresponding checkbox is automatically checked/unchecked.

4. The main form contains a **Show** button and the toolbar (the ToolStrip control) with 6 shortcut buttons (the ToolStripButton controls). The shortcut buttons on the toolbar have titles **1** – **6**, one of the shortcut buttons is in the pressed state (initially it is the shortcut button **1**). When the **Show** button is clicked, a second (non-modal) form appears containing 6 radio buttons (the RadioButton controls). Radio buttons are labeled **1** – **6**; the radio button corresponding to the pressed shortcut button in the main form should be selected. When another radio button is selected, the corresponding shortcut button is automatically pressed (and the previously pressed button is released); when the shortcut button is pressed, the corresponding radio button is automatically selected (and the previously pressed shortcut button is released).

5. The main form contains the **Show** button and six red labels with the text **Color**. When the **Show** button is clicked, a second (non-modal) form appears

containing six panels (the Panel controls); each panel contains three radio buttons. Radio buttons have labels **Red**, **Green**, **Blue** (these labels can be made common for all panels by placing three additional Label controls to the left of the first panel). Initially, the **Red** radio button is selected in each panel. When you switch radio buttons, the text color of the corresponding label on the main form is corrected. When you click on any label of the main form, its color changes cyclically (from red to green, from green to blue, from blue to red) and the corresponding radio button is automatically selected on the corresponding panel of the second form.

6. The main form contains the **Show** button and seven NumericUpDown controls with the text **0**. When the **Show** button is clicked, a second (non-modal) form with seven track bars (the TrackBar controls) appears. When you move the slider of some track bar, the number in the corresponding text box should automatically change. Specifying a different number in the text box should automatically change the slider position of the corresponding track bar. The range of values for track bars and text boxes is from 0 to 100. The track bar should only be synchronized with the text box if the correct number (0 to 100) is entered in the text box.

7. The main form contains seven progress bars (the ProgressBar controls), the **Default** button, and the **Show** button. When the **Show** button is clicked, a second (non-modal) form with seven track bars (the TrackBar controls) appears. As you change the slider position of some track bar, the content of the corresponding progress bar should automatically change (the range of values for track bars and progress bars is from 0 to 100). When you click on one of the progress bars, the corresponding track bar is toggled between available and unavailable state. When you click the **Default** button, all progress bars and track bars return to their initial (zero) position; the accessibility of the track bars does not change.

8. The main form contains the **Show** button and seven text boxes with the text **text1** – **text7**. When the **Show** button is clicked, a second (non-modal) form appears with seven track bars (the TrackBar controls) and labels containing the same text as the text boxes in the main form. When you change the slider position of some track bar, the width of the corresponding text box in the main form should automatically change (the range of values for track bars is from 0 to 100). When changing the text in some text box, the text of the corresponding label in the second form should automatically change.

9. The main form contains the **Show** button, seven track bars (the TrackBar controls), and seven labels with the text **label1** – **label7**. The range of values for track bars is from 10 to 30, the initial value is 10. When the **Show** button is clicked, a second (non-modal) form appears with seven text boxes. The content of the text boxes must be synchronized with the text of the labels of the main

form. When you change the slider position of some track bar, the font size for the corresponding text box changes according to the position of the slider (in the range from 10 to 30). When you change the text in some text box, the text in the corresponding label should change accordingly.

10. The main form contains the **Show** button and seven text boxes with the text **text1** – **text7**. When the **Show** button is clicked, a second (non-modal) form appears with seven text boxes that must be synchronized with the corresponding text boxes of the main form. Synchronization must be performed whenever the text in any text box changes. The second form also contains the **Stop Synchronization** button. When the **Stop Synchronization** button is clicked, synchronization stops and the button name is changed to **Resume Synchronization**. Clicking the button again resumes synchronization mode, and the text specified in the text boxes of the *main form* is used for synchronization. When you close and reopen the second form, the state of the **Stop/Resume Synchronization** button does not change.

11. The main form contains the **Show** button and seven text boxes with the text **text1** – **text7**. When the **Show** button is clicked, a second (non-modal) form appears with seven labels; the text of label coincides with the text of the corresponding text box. When you click on any label, the background color of the corresponding text box toggles between white and gray. When you change the text in the text box with a white background, the text of the corresponding label is changed accordingly; when you change the text in the text box with a gray background, the label does not change. However, when you change the background color of some text box from gray to white, the text of the text box and the text of the corresponding label should be immediately synchronized.

12. The main form contains the **Show** button and seven text boxes with the text **text1** – **text7**. When the **Show** button is clicked, a second (non-modal) form appears with seven track bars (the TrackBar controls); slider position of each track bar coinsides with the length of the text in the corresponding text box. When editing text in some text box, the slider position of the corresponding track bar is automatically changed; when the slider position of some track bar is changed, the text in the corresponding text box is shortened or lengthened (text lengthening is performed by adding * characters). Track bars can take values from 0 to 25; text longer than 25 characters cannot be input into the text box.

1.5. DRAGDROP project: drag-and-drop mode

General guidelines. In all projects, the form must have a fixed size. Drag-and-drop mode has been discussed in detail in the ZOO project (Chapter 10); in addition, it was used in the LISTBOXES (Section 19.4) and HTOWERS (Section 24.3) projects. Working with menus was considered in the TEXTEDIT1 and TEXTEDIT2 projects (Sections 12.1, 13.1–13.3).

1. The form contains three multi-line text boxes. Implement the ability to drag and drop text files from **Explorer** onto one of the text boxes; as a result of a such action, the contents of the text file is added to the existing text of this text box. Only files with the .**txt** extension should be processed. In addition, implement the ability to drag and drop the non-empty content of one text box onto another (while holding down the Ctrl key); as a result of a such action, the contents of the source text box is added to the previous contents of the target text box. The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command removes text from the text box that has focus.

2. The form contains three multi-line text boxes and three labels (each label is placed above the corresponding text box). Initially, the labels contain the text **<No file>**. Implement the ability to drag and drop text files from **Explorer** onto one of the *labels* and automatically load the contents of these files into the appropriate text box. Only files with the .**txt** extension should be processed; the full file name must be displayed in the corresponding label. You can only drag and drop a file onto the label with the text **<No file>**. The form menu contains one **Command** submenu with the **Save**, **Clear**, and **Exit** menu items. The **Save** and **Clear** commands affect the focus text box; the **Save** command saves the new contents of the text box in the same file, the **Clear** command clears the text box along with the associated label (the label displays the text **<No file>** again).

3. The form contains four "usual" labels with the letters of nucleotides **A**, **C**, **T**, **G** and three "wide" labels that have a frame and contain gene sequences (in the beginning, wide labels are empty). The width of wide labels does not depend on the size of the text and is determined by the width of the form. Implement the ability to drag and drop a nucleotide letter from a usual label onto a wide label; the nucleotides are added to the end of the text of the wide label. Also, implement the ability to drag and drop the contents of one wide label onto another; this action adds all text of the source label to the end of the target label text. The form menu contains one **Command** submenu with the **Clear 1**, **Clear 2**, **Clear 3**, and **Exit** menu items. The **Clear** commands clear the wide label with the specified number. All **Clear** commands must use one common handler.

4. The form contains six labels with the text **label1** – **label6** and six text boxes with the text **textBox1** – **textBox6**. When you drag and drop a text box onto a label (while holding down the Ctrl key), the text and font of the label changes to the text and font of the text box (the position of the text box does not change). You can drag and drop the text box onto the label several times. The form menu contains one **Command** submenu with the **Bold**, **Italic**, and **Exit** menu items. The **Bold** and **Italic** menu items act as checkboxes to set or unset the bold and italic font mode for the text box that has focus.

5. The form contains six buttons with titles **button1** – **button6** and six empty list boxes (the ListBox controls). When you drag and drop a button onto the list box, the title of this button is inserted *to the specified position* of the list (the position of the button does not change). You can drag and drop a button onto the list several times. The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command clears the contents of the list with focus; if a *button* has focus, the command performs no action. To determine the number of an item in the list by the position of the mouse cursor, use the IndexFromPoint method (see Section 19.4).

6. The form contains six radio buttons with titles **color1** – **color6** (titles have different colors) and six rectangles with a white background (use Panel controls as rectangles). When you drag and drop a radio button onto a rectangle, the background color of the rectangle changes to match the color of the title of the radio button being dragged (the position of the radio button does not change). The form menu contains one **Command** submenu with **Color** and **Exit** menu items. The **Color** command shows the ColorDialog dialog box to change the color of the title of the *selected* radio button. Working with the ColorDialog control is described in Section 13.3.

7. The form contains a panel (the Panel control) and two buttons with titles **New** and **Trash**. Pressing the **New** button creates a new label located in a random place on the panel; the label text is a random capital Latin letter. Implement the ability to drag and drop the appeared labels onto new location on the panel. Dragging a label onto the **Trash** button removes the label. When you drag and drop one label onto another, the source label is removed and the text of the source label is *added* to the text of the target label. The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command removes all labels. All labels are also removed by clicking the **Trash** button. See the HTOWERS project (Section 24.1) for information on creating controls at runtime.

8. The form contains four panels (Panel controls) and a group of four radio buttons with titles **1** – **4** for selecting the current panel. There are three labels with text **label1** – **label3** on *each* panel in random places. Implement the ability to drag and drop labels from the *current* panel to any other (in addition, for the current panel, you can drag and drop labels within this panel). Labels located on other panels cannot be dragged. The form menu contains one **Command** submenu with the **Color** and **Exit** menu items. The **Color** command shows the ColorDialog dialog box to change the color of all labels in the current panel. See the HTOWERS project (Section 24.3) for information on dragging and dropping controls between group controls. Working with the ColorDialog control is described in Section 13.3.

9. Implement an application to test drag-and-drop mode. The form contains four multi-line text boxes, a label with the text **Label**, and a label with the text **String**. Any of the labels, as well as any objects from other programs, in particular from **Explorer**, can be dragged and dropped onto any empty text box. As a result, detailed information about the drag object is displayed in the text box (including the available formats, which can be determined using the GetFormats method). When you drag the **Label** label, the drag object is the label itself; when you drag the **String** label, the drag object is the string with its name (that is, the string object). The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command clears the text box that has focus.

10. The form contains a list box for adding file names by dragging and dropping them from **Explorer** (the full file name is added to the *end* of the list). The form also contains the **Trash** label. When dragging the list onto the **Trash** label (while holding down the Ctrl key), the current list item is deleted and the next list item (or the previous one if the *last* list item was deleted) becomes the current one; the position of the list box does not change. If the list box is empty, then dragging it is not allowed (that is, the dragging cursor has a prohibition sign). The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command clears the list box. See the LISTBOXES project (Section 19.4) for information on dragging and dropping list items.

1.6. TIMER project: timer-controlled programs

General guidelines. In all projects, the form must be resizable; when changing the form size, the size and position of the form controls must be adjusted accordingly (using the Anchor or Dock properties). Working with a timer was discussed in the CLOCK project (Chapter 7).

1. The form contains a combo box with a list of comments (the ComboBox control with DropDownStyle = **DropDown**; initially, it contains only one list item "–"), an empty list box with the **Results** label, a *stopwatch label* with the text **0:0**, and a button with the title **Start** used to start the stopwatch (when the stopwatch starts, the button title changes to **Stop**).

The stopwatch displays seconds and tenths of a second. When you stop the stopwatch, its text, along with the current comment from the combo box, is appended to the end of the **Results** list and the stopwatch label is set to **0:0**. To add a new comment to the list of comments, just input it in the text field of the combo box and press Enter. The form menu contains one **Command** submenu with the **Clear** and **Exit** menu items. The **Clear** command clears the **Results** list.

2. The form contains a label with the lowercase Latin letter **a** and two read-only text boxes with the labels **Time** and **Points scored**. In the upper part of the form there is a toolbar with four shortcut buttons **Start**, **10 sec**, **30 sec**, **60 sec**; the last three shortcut buttons form a group that necessarily contains one button

in the pressed state (initially, it is the button with the title **10 sec**). In addition, the form contains a list box with the **Top Scores** label, this list box displays the top 10 scores for the selected time mode (the time mode is determined by the shortcut buttons **10 sec**, **30 sec**, **60 sec**). List of top scores is sorted in descending order.

When you press the **Start** button, the countdown begins in the **Time** text box (in tenths of a second), the **Points scored** text box is reset to zero, and the Latin letter in the label changes. It is required to press the key with the specified Latin letter. When the key is pressed correctly, the counter in the **Points scored** text box increases by 1 and the letter in the label changes again. If the key is pressed incorrectly, the **Points scored** counter decreases by 1. The duration of one training session (in seconds) is determined by the selected time mode, that is, by the shortcut button in the pressed state.

After the completion of the training session, the **Top Scores** list is corrected, if necessary; if the new score is added into the list, then this is reported in the auxiliary dialog box (use the MessageBox.Show function). The lists of top scores for each mode are stored in the files **scores10.dat**, **scores30.dat**, **scores60.dat** and are read from them when changing the mode and when starting the program.

3. The form contains a label and two read-only text boxes with the labels **Time** and **Points scored**. In the upper part of the form there is a toolbar with five shortcut buttons: **Start**, “+”, “–”, “*”, “/”; the last four shortcut buttons form a group that necessarily contains one button in the pressed state (initially, it is the button with the title “+”). In addition, the form contains a panel with five radio buttons with empty labels and a list box with the **Top Scores** label, this list box displays the top 10 scores for the selected math operation mode (the math operation mode is determined by the shortcut buttons “+”, “–”, “*”, “/”). List of top scores is sorted in descending order.

When you press the **Start** button, the countdown begins in the **Time** text box (in tenths of a second), the **Points scored** text box is reset to zero, and a numerical expression with the selected math operation appears in the label (for example, **34 + 78 =**). The group of radio buttons displays 5 answer options. You need to click on the radio button with the correct option. If the answer is correct, the counter in the **Points scored** text box is increased by 1; if the answer is incorrect, it is decreased by 1. In any case, a new expression appears in the label. The duration of one training session is 30 seconds.

After the completion of the training session, the **Top Scores** list is corrected, if necessary; if the new score is added into the list, then this is reported in the auxiliary dialog box (use the MessageBox.Show function). The lists of the top scores for each mode are stored in the files **add.dat**, **sub.dat**, **mult.dat**, **div.dat** and are read from them when changing the mode and when starting the program.

For the “+” and “–” modes, the expressions must use numbers from 1 to 100, for the “*” mode, the expressions must use numbers from 1 to 10, for

the "/" mode, the first operand must be two-digit number, the second one-digit number, and the result must be an integer.

4. The form contains a label with text displaying the current system time of the computer in the **hh:mm** format and three checkboxes associated with a separate alarm clock. The text near the checkbox indicates the alarm time and is also in the **hh: mm** format. Alarm clock is activated at the specified alarm time if the corresponding checkbox is checked; in this case, the checkbox changes state to **Indeterminate** and the sound signal (from the .**wav** sound file) is played for 10 seconds. If the sound file is less than 10 seconds long, the file is played in a loop (use the System.Media.SoundPlayer class to play .**wav** file).

To turn off the signal early, just uncheck the corresponding checkbox. After 10 seconds of sound signal playback, the checkbox is automatically unchecked. When you check any unchecked checkbox, a dialog box with two drop-down lists is displayed, in which you can set a new alarm time (hours and minutes); the default alarm time is the time previously associated with that alarm.

When the program finishes, it saves each alarm time and its current state in the **alarm.dat** text file. The saved data is restored when the program is started.

5. The form contains a read-only text box with the text **0:0** and the label **Time**, an empty list box with the label **Results**, and a panel that contains 6 small square labels numbered **1** – **6**. The form menu contains the **Command** submenu with the **Start** and **Exit** menu items.

When the **Start** command is executed, all the panel labels change their location on the panel randomly and are filled with a red backgroupd color, and the time count begins in the **Time** text box (in tenths of a second). You need to quickly click on all the panel labels in the ascending order of their numbers. Clicking on the correct label makes its background green. As soon as all 6 labels are clicked, the time count stops. If the **Time** text box contains a value less than **10:0** (that is, less than 10 seconds), then the message box with the text **You win!** is displayed and this time value is added to the top of the **Results** list box. Otherwise, the message box **You lost** is displayed.

The **Start** menu item should be available only when the game is stopped. The list of results must be stored in the **results.dat** file and read from this file each time the program is started. When changing the position of labels on the panel, it is necessary that labels do not intersect (see Comment 2 in Section 4.3).

6. The form contains a panel and two read-only text boxes with the labels **Time** and **Points scored**. In addition, the form contains a list box with the label **Top 5 scores**. Initially, the **Time** text box contains the number **30**; the **Points scored** text box contains the number **0**. The form menu contains one **Command** submenu with the **Start** and **Exit** menu items.

When the **Start** command is executed, the countdown begins in the **Time** text box (from 30 to 0, in seconds) and a small label with the number **50** appears

on the panel in a random place (the label size is 10 × 10 pixels). The number on the label decreases by 1 every tenth of a second. When you click on the label, the number in the **Points scored** text box increases by the number on the label (or decreases if the number on the label is negative) and the label is displayed elsewhere on the panel (again with the number **50**). When you click outside the label, the number **10** is subtracted from the **Points scored** value. After 30 seconds, the game ends. If the **Points scored** text box contains a positive number, then the message box with the text **You win!** is displayed, otherwise the message **You lost** is displayed; the list of the top scores is adjusted if necessary.

The **Start** menu item should be available only when the game is stopped. The top score list must be stored in the **score.dat** text file and read from this file each time the program is started.

7. The form contains a panel, a button with the title **Start**, three read-only text boxes with the labels **Accuracy**, **Time**, and **Result** (initially, the text boxes contain zeros), and a list box with the label **5 best results**.

When you click on the **Start** button, its title changes to **Stop**, the time count begins in the **Time** text box (in tenths of a second), and, in one of the corners of the panel, a square framed label without a text is displayed (the panel corner is selected randomly). It is required to drag this label with the mouse exactly to the center of the panel and press the **Stop** button (when dragging, the label must follow the mouse cursor). After that, the **Accuracy** text box displays two numbers: the horizontal and vertical deviations from the correct position (in pixels) and the **Result** text box displays a number calculated as follows: 1 is added to the time obtained (in seconds, with one fractional digit) and this number is multiplied by the sum of the absolute values of the deviations. See the MOUSE project (Section 9.1) for information on dragging with the mouse.

If the number in the **Result** text box is less than **50**, then the message box with the text **You win!** is displayed, otherwise the message **You lost** is displayed; the list of the best results is adjusted if necessary. The list of the best results must be stored in the **results.dat** text file and read from this file each time the program is started.

8. The form contains two read-only text boxes with the labels **Missiles** and **Time to explosion**, a panel, and a single-character label depicting an airplane (the **Wingdings** font, symbol **Q**) located in the left top corner of the panel. The mouse cursor on the panel looks like a cross. The form menu contains the **Command** submenu with the **Start** and **Exit** menu items.

When the **Start** command is executed, the airplane's label begins a straight-line movement on the panel (the increments of the `Left` and `Top` properties should be in the range 1–3; they are determined randomly before starting the game and are performed every tenth of a second) and the number **4** appears in the **Missiles** text box. Clicking on the panel marks the point of launching a missile, which

will explode after 2 seconds (in the **Time to explosion** text box, the countdown begins from **2.0** to **0.0**, in tenths of a second); the missile launch point on the screen should be marked with a red-colored label of the size 2 × 2 pixels. At the moment of the explosion, the size of the red-colored label increases to 30 × 30 pixels (the destruction area).

If, at the moment of the explosion, the airplane is in the destruction area, then the message box appears with the text **The airplane is shot down** and the game ends. Otherwise, nothing happens. You cannot launch a new missile before the explosion of a previously launched one. If all the missiles have been used without results or the airplane has reached the border of the panel, the message **You lost** is displayed. Use the IntersectsWith method to verify that the airplane is in the destruction area (see Comment 2 in Section 4.3).

The **Start** menu item should be available only when the game is stopped.

1.7. REGISTRY project: dialog boxes and working with the Windows registry

General guidelines. Working with the Windows registry is described in the IMGVIEW project (Sections 21.5–21.6). Working with the OpenFileDialog dialog box is described in the TEXTEDIT1 project (Section 12.3). To organize dialogs related to choosing a font and color, you should use the FontDialog and ColorDialog controls; an example of working with these controls is given in the TEXTEDIT2 project (Sections 13.3–13.4). Working with the SplitContainer control is described in the IMGVIEW project (Sections 21.1–21.2). Working with selections in the text boxes is discussed in the TEXTBOXES project (Section 8.1).

1. The form contains the SplitContainer control with a vertical splitter orientation. The left and right panels of the SplitContainer control contain one label and one multi-line text box. Initially, focus is on the left text box; pressing the Tab key toggles focus between the TextBox controls. When the form is resized, the panels of the SplitContainer control are sized proportionally (you cannot change the width of the panels by dragging the splitter; both panels always have the same width). The form menu contains one **File** submenu with the **Open...**, **Save**, **Compare**, and **Exit** menu items.

The **Open...** command displays the OpenFileDialog dialog box, which allows you to open existing text files, as well as create new ones (if the required file is missing, it is created automatically). When the required file is open, its text is loaded into the active text box and the full file name is displayed in the label above this text box.

The **Compare** command is available only if both TextBox controls contain loaded data; it compares the contents of the left and right text boxes and positions the cursor before the first differing character in the *left* text box (pressing

Tab should set the cursor before the first differing character in the *right* text box).

If the text of the text box is changed by the user, the symbol "*" is indicated in the label before the file name. The **Save** command saves the contents of the active text box in the file with the same name; after that, the symbol "*" disappears in the label.

At the end of the program, the file names are saved in the Windows registry; the next time the program is started, they are read from the registry; the size and position of the form and the position of the cursor in each text box should also be restored. When you try to close the program without saving the changed contents of the files, a standard dialog box appears asking if you want to save the changed file; the options are **Yes**, **No**, **Cancel**. If there are two unsaved files, two dialog boxes are displayed sequentially (unless you selected **Cancel** in the first dialog box).

2. The form contains a multi-line text box (the size of the text box is automatically resized when the form is resized), a drop-down list of options of number system conversion: **10 => 2** (this option is selected by default), **10 => 16**, **2 => 10**, **16 => 10**, **16 => 2**, **2 => 16**, and the **Convert** button. After the first launch of the program, at the beginning of its work, the text box is not available for editing. The form menu contains one **File** submenu with the **Open...**, **Save**, and **Exit** menu items.

The **Open...** command displays the OpenFileDialog dialog box, which allows you to open existing text files. As a result, the text of this file is loaded into the text box and the name of the loaded file is displayed in the form title bar.

The **Convert** button converts the number selected by the user in the text box from one number system to another (the selected number is replaced by the converted number; the converted number remains selected). If nothing is selected or the selection contains invalid data, then nothing happens.

If the text of the loaded file has been changed, the symbol "*" is displayed in the form title bar after the file name. The **Save** command saves the contents of the text box in the file with the same name; after that, the symbol "*" disappears in the form title bar.

At the end of the program, the name of the currently open file is saved in the Windows registry; the next time the program is started, it is read from the registry; the last conversion option, the size and position of the form, and the position of the cursor in the text box are also restored. When you try to close the program without saving the changed file contents, a standard dialog box appears asking if you want to save the changed file; the options are **Yes**, **No**, **Cancel**.

3. The form contains a list box with the **Playlist** label, the buttons **Play/Stop**, **Up** and **Down**, and the NumericUpDown control with the **Duration (sec)** label. The list box is automatically resized when the form is resized. If the

list box is empty, the buttons are inactive. The form menu contains one **Command** submenu with the **Add file**, **Add folder**, **Clear**, and **Exit** menu items.

The **Add file** and **Add folder** commands show the OpenFileDialog dialog box, which allows you to open existing wav-files; the **Add file** command adds the selected file to the list, the **Add folder** command adds *all* wav-files from the folder to the list (files are added to the end of the list). The last item added to the list becomes the current item. After adding at least one item to the list, the **Play/Stop** button becomes available; if the list contains more than one item, the **Up** and **Down** buttons become available.

Clicking the **Play/Stop** button starts playback of the current file from the list or stops playback of a file. The **Duration (sec)** counter allows you to specify the playback time of each file (in seconds); the default time is **10** seconds (if the file duration is less than the specified time, the file is played cyclically). The **Up** and **Down** buttons allow you to move the current item in the list up or down. When the playing time ends, the file located in the list after the current one starts playing automatically (this item of the list becomes the current one). Files are played cyclically. During playback, the ListBox control, the NumericUpDown control, and the **Up** and **Down** buttons are disabled. Use the System.Media.SoundPlayer class to play wav-files; working with list box items is described in the LISTBOXES project (Sections 19.2–19.3).

At the end of the program, the playlist, the position of the current list item, the playback time, the size and position of the form are saved in the Windows registry. The next time the program is started, it should restore the saved state.

4. The form contains the NumericUpDown control with the **Symbol code** label and a panel (the GroupBox control) with a label containing one character of 48 points size (the code of this symbol is specified in the NumericUpDown control). Initially, the program is configured for the **Wingdings** font; the name of the font is specified in the title of the GroupBox control. The form menu contains one **Command** submenu with the **Font...** and **Exit** menu items.

The **Font...** command shows the FontDialog dialog box, which allows you to change the name and style of the font, but not its size. The font size should be changed automatically when the form is resized; the initial form size cannot be reduced. When value of the NumericUpDown control changed, the corresponding symbol for the selected font is displayed in the panel. The displayed symbol must be centered vertically and horizontally relative to the border of the GroupBox control; to do this, set the appropriate values to the label properties AutoSize, Dock, TextAlign.

At the end of the program, the value of the NumericUpDown control, the name and style of the current font, as well as the size and position of the form are saved in the Windows registry. The next time the program is started, it should restore the saved state.

5. The form contains the NumericUpDown control with the **KnownColor number** label and a panel (the GroupBox control). When the form is resized, the panel size changes proportionally. When you input the number of one of the standard named colors from the KnownColor enumeration into the NumericUpDown control, the GroupBox panel is filled with this color and the name of this color is displayed in the title of this panel. The range of valid values for the NumericUpDown control must match the range of all standard named colors of the KnownColor enumeration: from **AliceBlue** to **YellowGreen**. The form menu contains one **Command** submenu with the **Color...** and **Exit** menu items.

The **Color...** command shows the ColorDialog dialog box, which allows you to select a color for the background of the GroupBox panel. If the selected color is one of the standard named colors, then its number appears in the NumericUpDown control, if the selected color is not a standard named color, then an error message is displayed in the standard message box and the panel background does not change. Working with colors is described in the COLORS and LISTBOXES projects (Chapter 18 and Section 19.1).

At the end of the program, the value of the NumericUpDown control and the size and position of the form are saved in the Windows registry. The next time the program is started, it should restore the saved state.

6. The form contains an empty multi-line text box (the TextBox control) unavailable for editing, with a gray background, the **Open** button, and 3 track bars (the TrackBar controls). Each track bar can take 10 values: from 0 to 9. The track bars are oriented vertically and are located in left-hand side of the form. A label is displayed above each track bar that contains the current value of that track bar (a number from **0** to **9**). The **Open** button is located under the track bars, the multi-line text box occupies the rest (right-hand) part of the form along its entire height.

When the form is resized, the width and height of the multi-line text box, as well as the height of the track bars, must change; the minimum allowable size of the form must be adjusted so that they provide the display of all its controls.

When you set the correct three-digit *lock code* with the track bars and then click the **Open** button (or press Enter), the content of the **notebook.txt** file (if this file exists) is loaded into the TextBox control, the background of the TextBox control turns white, the text box becomes editable, and the button title changes to **Close** (if the code is set incorrectly, the appearance of the TextBox control and the button does not change).

The correct lock code is stored in the Windows registry; if there is no the corresponding subkey in the registry, then the code is **000**. When the text box is editable, you can set new lock code using the track bars.

When you click the **Close** button, the text is saved in the **notebook.txt** file, the code is saved in the Windows registry, the text box is cleared, its background

is grayed out, the button title is changed to **Open**, and the values of track bars are changed randomly.

At the end of the program, the size and position of the form are additionally saved in the Windows registry and are restored the next time the program is started.

7. The form contains the SplitContainer control with a vertical splitter orientation. The list box is located on the left panel of the SplitContainer control, a multi-line text box is located on the right panel. The list box and text box are automatically resized when the form is resized; when the form width changes, the width of the *text box* changes. The splitter between the left and right panel of the SplitContainer control can be dragged. The Tab key allows you to toggle focus between the list box and the text box. The form menu contains one **File** submenu with the **Open...**, **Close**, and **Exit** menu items.

The **Open...** command displays the OpenFileDialog dialog box which allows you to open existing text files, as well as create new ones (if the required file is missing, it is created automatically). When the required file is open, its full name is added to the end of the list box (and becomes the selected list item) and its text is loaded into the text box. In the future, to load this file into the text box, it is enough to select its name in the list box. If, when executing the **Open...** command, the name of the file is already included in the list box, then this name in the list is made selected. When a list item loses selection, its associated text is automatically saved in the appropriate file.

The **Close** command removes the name of the selected file from the list of files; this action also automatically saves the text in the file. When a list item is deleted, the next item is selected; if the next item is absent, the previous item is selected. If the list box is empty, then the text box is not editable and the **Close** menu item is unavailable. Working with list box items is described in the LISTBOXES project (Sections 19.2–19.3).

At the end of the program, the file list is saved in the Windows registry. The index of the selected list item, the position of the cursor in the text box, the size and position of the form, the position of the splitter between the left and right panels are also saved in the registry. The next time the program is started, it should restore the saved state.

1.8. MDIFORMS project: MDI applications

Working with MDI applications is described in the JPEGVIEW project (Chapter 22).

1. The MDI main form initially contains one special child form with the title **Clipboard**, which is entirely occupied by the multi-line TextBox control. This child form acts as the application's own *clipboard*. The MDI application menu includes three submenus: **File** (the **Open** and **Exit** menu items), **Clipboard** (the

Cut, **Copy**, **Paste** menu items), and **Window** (the **HTile**, **VTile**, **Cascade**, **Arrange Icons** menu items, as well as a list of child forms). The commands of the **Window** submenu provide standard MDI application actions related to the placement of child forms and their selection (see Section 22.2).

The **Open** command allows you to create or load a text file and display it in a new child form with the TextBox control (in this case, the **Close** menu item appears in the **File** submenu; this command closes the active child form). In the OpenFileDialog control used to select the file name, you should set the file mask (the Filter property) to display only text files (with the .**txt** extension). If a file with the specified name does not exist, then it is created. The full name of the created or loaded file is displayed in the title bar of the corresponding child form; when you try to reload an existing file, a new child form is not created; instead, the child form that already contains the specified file becomes active. When the child form is closed, the corresponding text file is automatically saved.

When executing the commands **Cut**, **Copy**, **Paste**, the **TextBox** control of the **Clipboard** child form should be used (instead of the standard Windows clipboard): the text cut or copied from any other child form should be placed on the **Clipboard** child form (its previous content is deleted). When the **Paste** command is executed, the text from the **Clipboard** form should be inserted into the current position of the active child form. The contents of the **Clipboard** form can be edited, however, commands related to copying, cutting and pasting cannot be executed for it; furthermore, this child form cannot be closed.

2. The MDI main form initially contains no child forms. The MDI application menu includes two submenus: **File** (the **Open** and **Exit** menu items) and **Group** (the **Open Group** menu item).

The **Open** command allows you to create or load a text file and display it in a new child form with the TextBox control; the **Open Group** command allows you to immediately load all text files from the selected directory. In the OpenFileDialog control used to select the file name, you should set the file mask (the Filter property) to display only text files (with the .**txt** extension). The OpenFileDialog control is also used for group file loading; when the **Open Group** command is executed, it is sufficient to select *one* of the text files in the required directory to load *all* text files from this directory. When executing the **Open** command, you can specify the name of a non-existent file; in this case, it is created. The full name of the created or loaded file is displayed in the title bar of the corresponding child form; when you try to reload an existing file, a new child form is not created; instead, the child form that already contains the specified file becomes active. A similar condition must be satisfied for a group loading: already loaded files are not reloaded.

If there is at least one child form, the **Close** menu item appears in the **File** submenu (this command closes the active child form) and the **Close Group** and

Close All menu items appear in the **Group** submenu (the **Close Group** command closes the active child form and all other child forms with files from the same directory as the active child form file; the **Close All** command closes all child forms). When the child form is closed, the corresponding text file is automatically saved.

In addition, if there is at least one child form, the **Window** submenu appears in the application menu with the **HTile**, **VTile**, **Cascade**, **Arrange Icons** menu items, as well as with a list of child forms. The commands of the **Window** submenu provide standard MDI application actions related to the placement of child forms and their selection (see Section 22.2).

3. The MDI main form initially contains no child forms. The MDI application menu includes two submenus: **File** (the **Open** and **Exit** menu items) and **Actions** (the **Shift Forward**, **Shift Backward**, and **Union** menu items; these menu items are available only if there are at least *two* child forms).

The **Open** command allows you to create or load a text file and display it in a new child form with a TextBox control. In the OpenFileDialog control used to select the file name, you should set the file mask (the Filter property) to display only text files (with the .**txt** extension). When executing the **Open** command, you can specify the name of a non-existent file; in this case, it is created. The full name of the created or loaded file is displayed in the title bar of the corresponding child form; when you try to reload an existing file, a new child form is not created; instead, the child form that already contains the specified file becomes active. If there is at least one child form, the **File** submenu displays the **Close** menu item that closes the active child form, and the **Close All** menu item that closes all child forms. When the child form is closed, the corresponding text file is automatically saved.

The **Shift Forward**, **Shift Backward**, and **Union** commands change the contents of the child forms as follows. The **Shift Forward** command performs a *cyclic shift forward*, that is, it moves the contents of the first child form to the second child form, the contents of the second child form to the third child form, …, the contents of the last child form to the first child form. The **Shift Backward** command performs a *cyclic shift backward*, that is, it moves the contents of the second child form to the first child form, the contents of the third child form to the second child form, …, the contents of the first child form to the last child form. The **Union** command combines the contents of all child forms into the active child form; the text is added in the order of child form numbers, starting with the active child form (for example, when executing the **Union** command for the third of five loaded forms, the initial text of the child forms with the following numbers will be written in the third form: 3, 4, 5, 1, 2).

If there is at least one child form, the **Window** submenu appears in the application menu with the **HTile**, **VTile**, **Cascade**, **Arrange Icons** menu items, as well as with a list of child forms. The commands of the **Window** submenu pro-

vide standard MDI application actions related to the placement of child forms and their selection (see Section 22.2).

4. The MDI main form initially contains no child forms. The MDI application menu includes two submenus: **File** (the **Open** and **Exit** menu items) and **Actions** (the **Move**, **Add**, and **Swap** menu items; these menu items are available only if there are at least *two* child forms).

The **Open** command allows you to create or load a text file and display it in a new child form with a TextBox control. In the OpenFileDialog control used to select the file name, you should set the file mask (the Filter property) to display only text files (with the .**txt** extension). When executing the **Open** command, you can specify the name of a non-existent file; in this case, it is created. The full name of the created or loaded file is displayed in the title bar of the corresponding child form; when you try to reload an existing file, a new child form is not created; instead, the child form that already contains the specified file becomes active. If there is at least one child form, the **File** submenu displays the **Close** menu item that closes the active child form, and the **Close All** menu item that closes all child forms. When the child form is closed, the corresponding text file is automatically saved.

The **Move** command changes the order of the child forms by moving the active form to the end of the list of child forms. To implement this command, it is enough to close the active child form (and save, if necessary, its contents in the corresponding file) and create a new child form with the same contents.

The **Add** and **Swap** commands modify the contents of the child forms. The **Add** command adds the contents of the form that *follows* the active form to the contents of the active form. The **Swap** command swaps the contents of the active child form and the form that follows the active form. The first child form is assumed to follow the last child form.

If there is at least one child form, the Window submenu appears in the application menu with the **HTile**, **VTile**, **Cascade**, **Arrange Icons** menu items, as well as with a list of child forms. The commands of the **Window** submenu provide standard MDI application actions related to the placement of child forms and their selection (see Section 22.2).