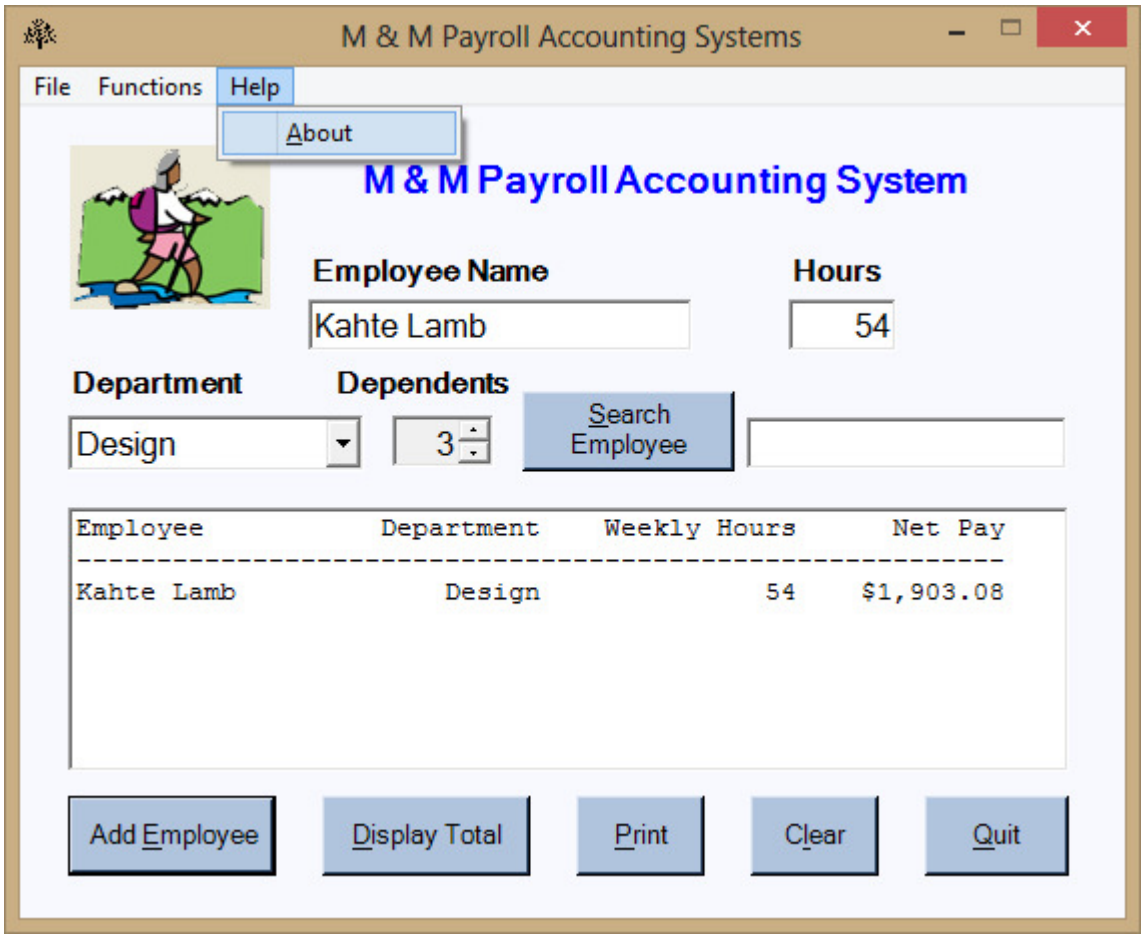
DUE: 12/2/2013

Project #3—M & M Crafters

Employees at M &M Crafters are grouped in the following five departments with respective hourly wages per hour: Sales employees get $15 per hour, Production employees get $25 per hour, Design employees get $40 per hour, Shipping employees get $10 per hour, and Other employees get $12 per hour. These hourly rates are for a regular 40-hours workweek and all employees are paid time and a half for overtime. Withholdings from the gross pay include federal tax = 20% * (gross – dependents * 48.60) and state tax = 3.2% * gross pay. Net weekly pay is the gross pay minus the federal and state taxes.

Write a C# program that allows the user to enter the employees name, weekly hours worked, and number of dependents that the employees has, and then calculates and displays in a ListBox the net pay for the employee based on which department he/she belongs (selected from a department ComboBox and populated from a text file at form_load).



Your program must contain/do:

1. Add Employee button which adds the employee's information (employee's name, department, weekly hours worked, and net pay for the week) into the display ListBox. Note that it is in this procedure that the net pay calculations are performed and displayed into the ListBox.
2. A procedure which executes when the form loads. In this event procedure you will load, at run-time, at least the five departments into the Department ComboBox. The departments are to be read from an external file, Departments.txt, that may contain unknown number of departments. You should not allow the user to enter his/her own department in the combo box. Consider doing the same for the employees.
3. Avoid displaying the same employees in the list box or multiple entries of the total. Consider using a boolean function to check existence of employee entry in the list box in order not to allow duplicating.
4. Header, description of the program, and comments.
5. Menu with at least two main functional menu choices.
6. At least one function—I recommend a function that calculates the net pay.
7. Search for specified employee capability. This means that the user inputs the name of the employee, and the employee's name, department, weekly hours worked, and net pay for the week are displayed in the same list box (or a message that that employee is not found).
8. Add Employee, Calculate Total, Search, Clear, and Quit buttons.
9. Validate input for proper type and range.
10. An About form and a Splash screen.
11. Printing capability to print the content of the listbox and print the form.

EXTRA CREDIT (Optional). For 3 additional points provide ability to save the content of the listbox, read the employees and the department in which they belong an external text file and populate an employee name combobox, and ability to delete a selected employee from the list. There are many other improvements one could make for this project. Consider, for example,

- Use an XML, Excel, MS Access or SQL database to read employees' names and their corresponding departments.
- Double-click on an employee item in the ListBox in order to delete that record (after a confirmation). This will require updating the total.
- Change the employee name textbox to a combo box and read and populate it from an external Employees.txt file.
- Search with wild characters (* and ?) so that searching for a name Ni* should list all employees whose name begins with Ni followed by any number of other letters (Nikki, Nicholas, Niko, Nivena, Niprumash, etc.). The wild character ? replaces any one single character.

- Allow user to open a file to read various Departments (use OpenFileDialog). Consider using a Dictionary structure for various departments and hourly rates.
- Create an Employee class that has as private instance variables name, department, dependents, and hours worked. Read the Employees data from a external file and create an array of Employee objects. Use this array for searching, displaying, etc.

Project Guidelines

Problem Statement: The following list reviews the GUI design and programming guidelines you have learned so far. You can use this list to verify that the applications and projects you create adhere to the standards outlined herein.

- Information should flow either vertically or horizontally, with the most important information always located in the upper-left corner of the screen.
- Maintain a consistent margin of two or three dots from the edge of the window.
- Related controls should be grouped together using either white space or a group box.
- Position related controls on succeeding dots.
- Buttons should either be centered along the bottom of the screen or stacked in either the upper-right or lower-right corner.
- If the buttons are centered along the bottom of the screen, then each button should be the same height; their widths, however, may vary.
- If the buttons are stacked in either the upper-right or lower-right corner of the screen, then each button should be the same height and the same width.
- Use no more than six buttons on a screen.
- The most commonly used button should be placed first.
- Button captions should:
  - ✓ be meaningful
  - ✓ be from one to three words appear on one line
  - ✓ be entered using book title capitalization
  - ✓ contain & for “hot” keys activation
- Use labels to identify the controls in the interface.
- Identifying labels should:
  - ✓ be from one to three words
  - ✓ appear on one line
  - ✓ be aligned by their left borders
  - ✓ be positioned either above or to the left of the control that they identify
  - ✓ end with a colon (:)
  - ✓ be entered using sentence capitalization
  - ✓ have their BackStyle property set to 0-Transparent
  - ✓ have their BorderStyle property set to 0-None
- Labels that display program output (for example, the results of calculations):
  - ✓ should have their BorderStyle set to 1-Fixed Single
  - ✓ should have their Appearance property set to 0-Flat
  - ✓ can have their BackStyle property set to either 0-Transparent or 1-Opaque
- Align labels and controls to minimize the number of different margins.

- If you use a graphic in the interface, use a small one and place it in a location that will not distract the user.
- Use no more than two different font sizes, which should be 10, 12 or 14 points.
- Use only one font type, which should be a sans serif font, in the interface.
- Avoid using italics and underlining.
- Use light colors such as white, off-white, light gray, pale blue, or pale yellow for an application's background, and very dark color such as black for the text. Alternately, use dark colors for the form's background with light colors for the text—contrast is the name of the game.
- Use color sparingly and don't use it as the only means of identification for an element in the interface.
- Set each control's TabIndex property to a number that represents the order in which you want that control to receive the focus (begin with 0).
- A text box's TabIndex value should be one more than the TabIndex value of its identifying label.
- Lock the controls in place on the form.
- Assign a unique access key to each essential element of the interface (text boxes, buttons, and so on).
- Document the program internally.
- Enter the Option Explicit and Option Strict statements in the General Declaration section of every form and module.
- Use variables to control the preciseness of numbers in the interface.
- Use the Val function on Text and Caption properties, and when assigning the result of the InputBox function to a numeric variable.
- Set the form's BorderStyle, MinButton, MaxButton, and ControlBox properties appropriately:
  - ✓ Splash screens should not have a Minimize, Maximize, or Close button, and their borders should not be sizable.
  - ✓ Forms that are acting as dialog boxes should have a BorderStyle property of 3-Fixed Dialog.
  - ✓ Forms other than splash screens and dialog boxes should always have a Minimize button and a Close button, but disable the Maximize button.
  - ✓ Forms other than splash screens and dialog boxes typically have a BorderStyle property setting of either 1-Fixed Single or 2-Sizable.
- Test the application with both valid and invalid data (test the application without entering any data; also test it by entering letters where numbers are expected). Test the application for correct results with known data.
- In the InputBox function, use sentence capitalization for the prompt, and book title capitalization for the title.
- Center a form by including the appropriate formulas in the form's Load event.
- Display an hourglass or an arrow and an hourglass if a procedure will take a long period of time.
- Remove excess or duplicated code from the application.

Grading Guidelines: Your project will be graded on the following:

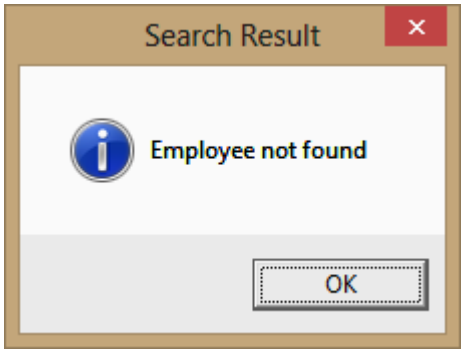
- Correct solution to the proposed problem.
- GUI design (as outlined above).
- Elegance, simplicity, style, and readability of the algorithms and code.
- Comments:

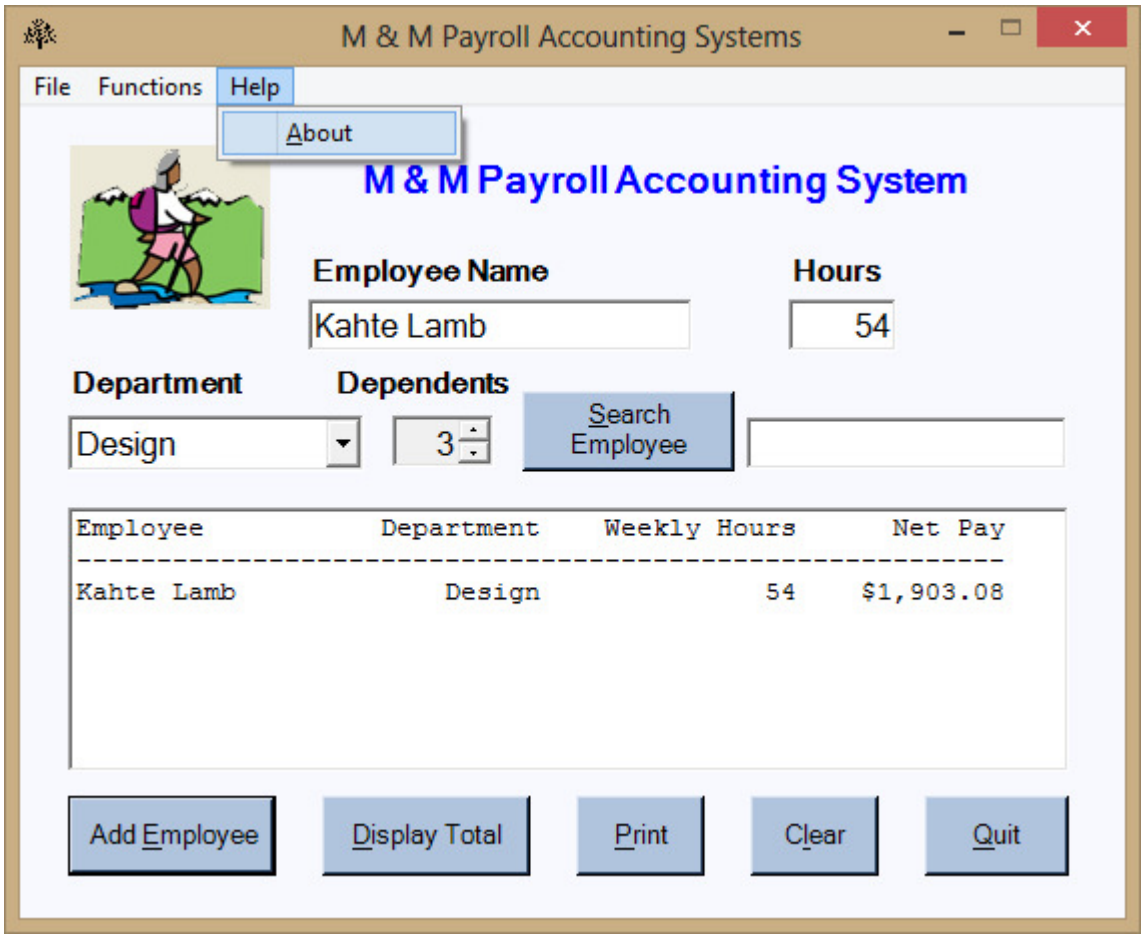
- ✓ Use of header.
- ✓ Description of project.
- ✓ Description of procedures.
- ✓ Description of complicated code.

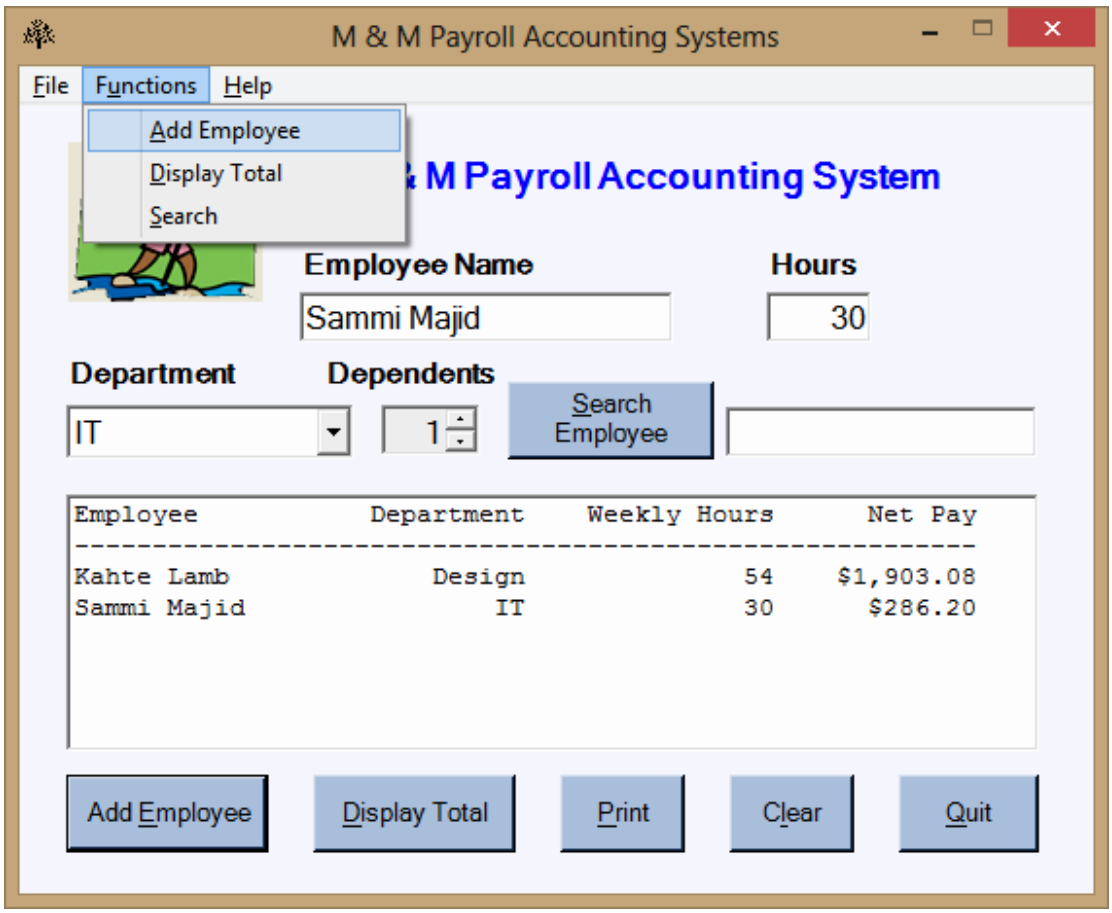
Hints for Project #3: *Given that your GUI resembles the given one, follow these guidelines:*

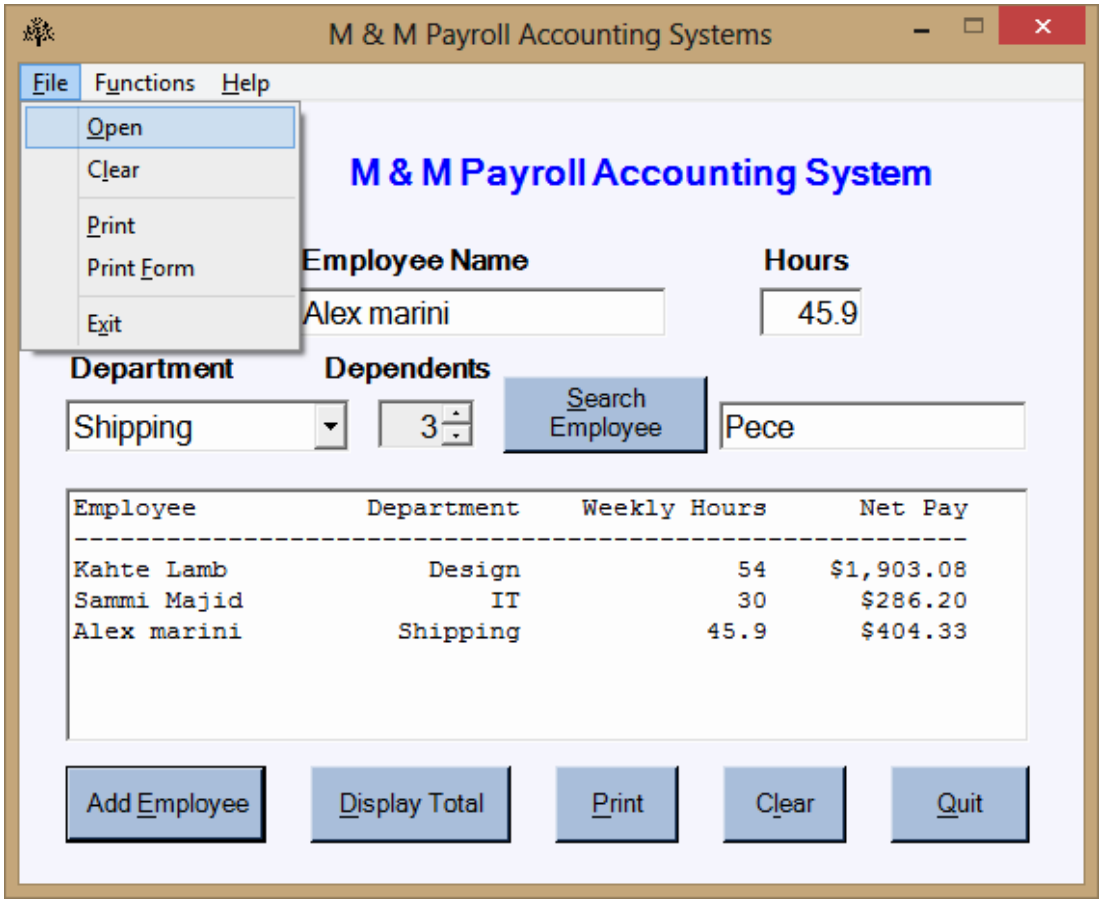
- In the frmPayroll_Load event procedure add the contents of the Departments.txt file to the ComboBox using a Do While. The StreamReader's method Peek returns -1 when you read end of file. Alternatively, read from employees names and corresponding departments from an XML, Excel, or MS Access file.
- Consider disabling the buttons Total, Clear, and Search in the same frmPayroll_Load event procedure initially to prevent their operations on empty ListBox.
- Do not allow duplication of entries in the list box--this means that only one total and no duplicate employees show up in the output listbox.
- Declare a class level variable decTotal variable to keep track of all payments; it accumulates when new employee is added to the ListBox.
- The btnAddEmployee event procedure should read the entered employee data (from the text boxes), calculate the net pay by calling a CalculatePay function, increase the decTotal, and display the record in the ListBox. It should also enable the three buttons that were disabled in the frmPayroll_Load event procedure.
- Calculate the net pay by your own function that implements the same process as used in Lab 2: check for overtime, calculate gross pay, federal and state tax, and then calculate net pay. Consider using a switch statement in order to use the appropriate hourly rate. For testing purpose, change one of the constant rates to $28 per hour in order to check your net pay calculation as shown in Lab 2.
- The btnTotal should just add the total of all employees net payments to the ListBox.
- The btnClear should clear all text boxes, restore the header in the ListBox, and disable the Total, Search and Clear buttons. It should also reset the decTotal variable to 0 and set the focus to the employee's name TextBox.
- On non-empty search string the btnSearch button should cycle through (read a loop here) the ListBox's items and find and display the first match by comparing only the name portion of the ListBox's items. Consider using a Boolean variable blnFound initially False but set to True if there is a match and a counter variable intCounter initially set to 0 that is incremented each time through the loop up to lstDisplay.Count - 1 of the ListBox. Note that the Items collection of the ListBox is like an array that is zero-based; thus, if the ListBox has 5 items, they are accessible with indices 0-4, i.e. lstDisplay.Items.Item(0), lstDisplay.Items.Item(1), …, lstDisplay.Items.Item(4). If there a match with the search name, display only that record in the ListBox (or highlight it); otherwise display Record Not Found message in a MessageBox.

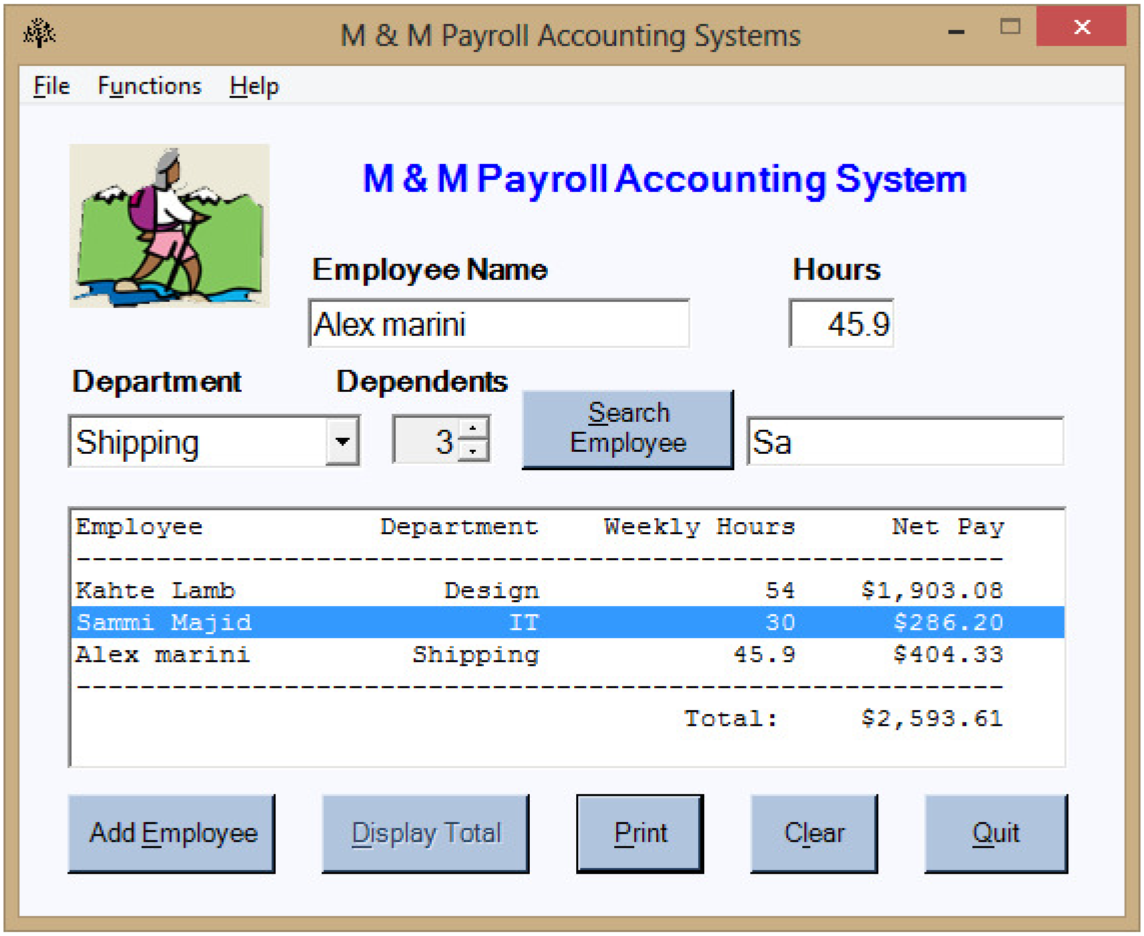
- Consider using the substring method to implement the extra credit portion of the search for an employee when the name does not match exactly or is given only partially.
- Although not required, consider using a FileOpenDialog control to allow the user to specify which file to use in order to read new departments into the ComboBox (and replace the old ones).
- The File menu should have Open, Clear, Separator, Print, Separator, and Exit choices. The Actions menu should have Add Employee, Display Total and Search choices. The Help menu should have an About choice for an about form. Other than the Open and About, the menu selection duplicate the buttons and should call those event procedures—do not duplicate code. A screen capture of the Actions menu and initial state (before any records are added) follows. The two others that follow are showing a successful Search. You may wish to report an unsuccessful search with a MessageBox.











Pay attention to the following criteria for grading:

'~~~~~~

'Great job. Here are some ways to improve it:

'1. Include more comments, especially after each procedure heading.

'2. Use functions, and subroutines to divide program into modules.

'3. Check your calculations--50 hours for Design with 3 dependents should give $1718.76.

'4. Disable all buttons initially except Quit and Add Employee. Enable and disable buttons and menus according to appropriate functionality.

'5. Add an image to the form.

'6. Use constant declaration for pay rates, tax rates and other constants.

'7. Add departments into combo box at form_load time.

'8. Do not allow user to enter different department—make combo box drop-down list.

'9. Validate inputs for proper type and range.

'10. Avoid menu choices that are not implemented.

'11. Follow C# naming convention.

'12. Add icon to form.

'13. Avoid excessive use of class-level variables.

'14. Missing menus.

'15. Add Splash screen and About form.

'16. Clear should clear text boxes, list box, reset total to 0, and set focus to name text box.

'17. Print not implemented. Need two prints: form and content of listbox.

'18. Make Add Employee button default.

'19. Search employee by first name or last name.
'20. Search does not work.
'20. Disable form from maximization.
'21.* Allow user to search with partial name.
'22. Right-justify net pay in list box.
'23. Select a default department in the combo box.
'24. Consider moving the overtime check in a function.
'25. Clear combo box before adding departments into it.
'26. Good effort in using a separate class for Employee, but save as a separate file.
'You are not following conventions for setter and getter methods of classes.
'27. Total not working. btnDisplayTotal should be two lines : display a line of dashes (---) and decTotal (declared as a class level variable but updated in btnAddEmployee_Click).
'28. Display message when employee is not found.
'29. Name project appropriately--Project 3 won't do.
'30. Place focus to offending text box when validating input.
'31. Display totals only once.
'32. Synchronize menu choices with button--if a button is disabled, for example,
'so should the corresponding menu choice.
'33. Use NumericUpdown for number of dependents.
'34. Missing ToolTip.
'
'The ones that apply to you are:
'9, 15, 20, 23, 32
'
'26+1=27/30
'~~~~~~