2. Imagine we have the list shown in Figure 5-26 implemented as a linked list. As discussed in "List Search," in Section 5.2, the search needs to be able to pass back both the location of the predecessor (`pPre`) and the location of the current (`pLoc`) node based on search criteria.

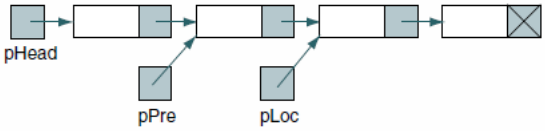

FIGURE 5-26 Linked List Implementation for Exercise 2

The following code to set `pPre` and `pLoc` contains a common error. What is it and how should it be corrected?

```
pLoc = pLoc->link
pPre = pPre->link
```

(*Hint:* What are the contents of these pointers at the beginning of the search?)

3. Imagine we implement a list using a dummy node at the beginning of the list. The dummy node does not carry any data. It is not the first data node, it is an empty node. Figure 5-27 shows a list with a dummy node.

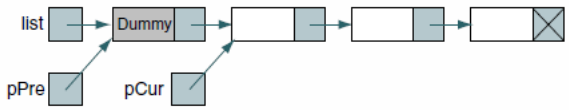

FIGURE 5-27 Linked List Implementation for Exercise 3

Write the code to delete the first node (the node after the dummy node) in the list.

4. Write the code to delete a node in the middle of a list implemented as a linked list with the dummy node (see Exercise 3). Compare your answer with the answer to Exercise 3. Are they the same? What do you conclude? Does the dummy node simplify the operation on a list? How?

7. Imagine we have the two lists shown in Figure 5-29. What would happen if we applied the following statement to these two lists?

```
listl = list2
```

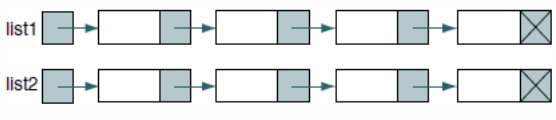

FIGURE 5-29 Linked Lists for Exercise 7

26. Write a program to read a list of students from a file and create a list. The program should use a linked list for implementation. Each node in the linked list should have the student's name, a pointer to the next student, and a pointer to a linked list of scores. There may be up to four scores for each student. The structure is shown in Figure 5-33.

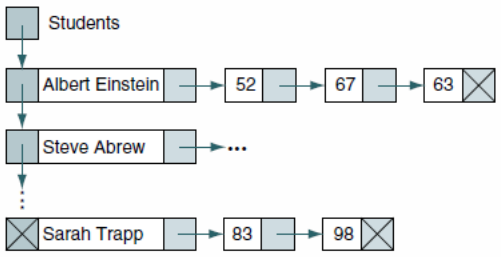

FIGURE 5-33 Data Structure for Project 26

The program should initialize the student list by reading the students' names from the text file and creating null score lists. It should then loop through the list, prompting the user to enter the scores for each student. The scores' prompt should include the name of the student.

After all scores have been entered, the program should print the scores for each student along with the score total and the average score. The average should include only those scores present.

The data for each student are shown in Table 5-3.

Student name	Score 1	Score 2	Score 3	Score 4
Albert Einstein	52	67	63	
Steve Abrew	90	86	90	93
David Nagasake	100	85	93	89
Mike Black	81	87	81	85
Andrew Dijkstra	90	82	95	87
Joanne Nguyen	84	80	95	91
Chris Walljasper	86	100	96	89
Fred Albert	70	68		
Dennis Dudley	74	79	77	81
Leo Rice	95			
Fred Flintstone	73	81	78	74
Frances Dupre	82	76	79	
Dave Light	89	76	91	83
Hua Tran	91	81	87	94
Sarah Trapp	83	98	94	93

TABLE 5-3 Data for Project 26

29. Write a program that adds and subtracts polynomials. Each polynomial should be represented as a list with linked list implementation. The first node in the list represents the first term in the polynomial, the second node represents the second term, and so forth.

Each node contains three fields. The first field is the term's coefficient. The second field is the term's power, and the third field is a pointer to the next term. For example, consider the polynomials shown in Figure 5-34. The first term in the first polynomial has a coefficient of 5 and an exponent of 4, which is then interpreted as $5x^4$.

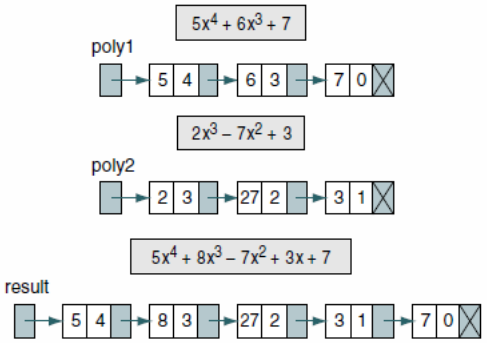

FIGURE 5-34 Example of Polynomials for Project 29

The rules for the addition of polynomials are as follows:

a. If the powers are equal, the coefficients are algebraically added.
b. If the powers are unequal, the term with the higher power is inserted in the new polynomial.
c. If the exponent is 0, it represents x^0, which is 1. The value of the term is therefore the value of the coefficient.
d. If the result of adding the coefficients results in 0, the term is dropped (0 times anything is 0).

A polynomial is represented by a series of lines, each of which has two integers. The first integer represents the coefficient; the second integer represents the exponent. Thus, the first polynomial in Figure 5-35 is

```
5    4
6    3
7    0
```

To add two polynomials, the program reads the coefficients and exponents for each polynomial and places them into a linked list. The input can be read from separate files or entered from the keyboard with appropriate user prompts. After the polynomials have been stored, they are added and the results are placed in a third linked list.

The polynomials are added using an operational merge process. An operational merge combines the two lists while performing one or more operations—in our case, addition. To add we take one term from each of the polynomials and compare the exponents. If the two exponents are equal, the coefficients are added to create a new coefficient. If the new coefficient is 0, the term is dropped; if it is not 0, it is appended to the linked list for the resulting polynomial. If one of the exponents is larger than the other, the corresponding term is immediately placed into the new linked list, and the term with the smaller exponent is held to be compared with the next term from the other list. If one list ends before the other, the rest of the longer list is simply appended to the list for the new polynomial.

Print the two input polynomials and their sum by traversing the linked lists and displaying them as sets of numbers. Be sure to label each polynomial.

Test your program with the two polynomials shown in Table 5-4.

Polynomial 1		Polynomial 2	
Coefficient	**Exponent**	**Coefficient**	**Exponent**
7	9	–7	9
2	6	2	8
3	5	–5	7
4	4	2	4
2	3	2	3
6	2	9	2
6	0	–7	1

TABLE 5-4 Text Data for Project 29

30. In older personal computers, the largest integer is 32,767 and the largest long integer is 2,147,483,647. Some applications, such as cryptography and security algorithms, may require an unbounded integer. One way to store and manipulate integers of unlimited size is by using a linked list. Each digit is stored in a node of the list. For example, Figure 5-35 shows how we could store a five-digit number in a list.

Although the list in Figure 5-35 is represented as moving from right to left, there is no physical direction in a list. We represent it in this way to clarify the problem.

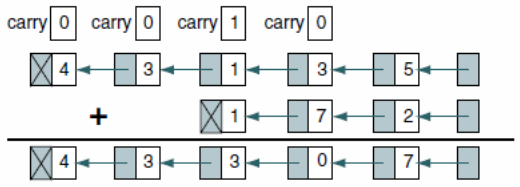

FIGURE 5-35 Integer Stored in a List for Project 30

To add two numbers, we simply add the corresponding digit in the same location in their respective lists with the carry from the previous addition. With each addition, if the sum is greater than 10, we need to subtract 10 and set the carry to 1. Otherwise, the carry is set to 0.

Write an algorithm to add two integer lists. Design your solution so that the same logic adds the first numbers (units position) as well as the rest of the number. In other words, do not have special one-time logic for adding the units position.

34. Write a program to process stock data. The stock data should be read from a text file containing the following data: stock code, stock name, amount invested (*xxx.xx*), shares held, and current price. Use the Internet or your local paper to gather data on at least 20 stocks. (You may use mutual funds in place of stocks.)

    As each stock is read, insert it into a doubly linked multilinked list. The first logical list should be ordered on the stock code. The second logical list should be ordered on the gain or loss for the stock. Gain or loss is calculated as the current value of the stock (shares held times current price) minus the amount invested. Include at least one loss in your test data.

    After building the lists, display a menu that allows the user to display each logical list forward or backward (a total of four options). Each display should contain an appropriate heading and column captions.

    Run your program and submit a list of your input data and a printout of each display option.