**Chapter 4 Linked-Based Implementations - Answers**

Top of Form

Bottom of Form

**Content**

**Answers**

**Important note:** When you are writing application code to use a class template, you must specify the actual data type that you want to use in the class template. For example:

    Node<double> \*myPtr;

When you are writing code inside a template where ItemType is the name that represents the type of data that is being processed, you need:

    Node<ItemType> \*myPtr;

In question 1 below, you are asked to write application code.

**1. Using the Node class template from Chapter 4, this code will create a linked list with 3 nodes:**

**// Create an empty list with pointers to first and last node**  
**Node <int> \*first = nullptr;    // create head pointer for list**   
**Node <int> \*last = nullptr;    // pointer to last node in list**   
  
**Node<int> \*temp;            // temporary pointer**  
  
**first = new Node<int>(2);            // add first node to list**  
**last = first;**  
  
**temp = new Node<int>(4);    // create a second node**   
**last->setNext(temp);        // and add it to end of list**  
**last = last->getNext();**  
  
**temp = new Node<int>(6);    // create a third node**   
**last->setNext(temp);        // and add it to end of list**  
**last = last->getNext();**

**You might want to try creating a small program using this code and try it for yourself. It would be useful to write a function to display the contents of the list.**

**Write code to remove the second node from the list. Be sure to free the memory occupied by the freed node, and that the linked list still contains the remaining nodes.**

Code to remove the second node:

    temp = first->getNext();            //make temp point to node to be removed      
    first->setNext(temp->getNext());    // make first node point to third node  
    delete temp;                        // free removed node

Code to display the items in the linked list (putting them in a function is left as an exercise):

    temp = first;  
    while (temp != nullptr)  
    {  
        cout << temp->getItem() << ' ';  
        temp = temp->getNext();  
    }

**2. The getFrequencyOf method in the LinkedBag class template has a loop that travereses teh linked list that holds the bag entries:**

**int frequency = 0;  
   int counter = 0;  
   Node<ItemType>\* curPtr = headPtr;  
   while ((curPtr != nullptr) && (counter < itemCount))  
   {  
      if (anEntry == curPtr->getItem())  
      {  
         frequency++;  
      } // end if  
        
      counter++;  
      curPtr = curPtr->getNext();  
   } // end while**

**This method had some redundancy build into it. It checks both the count of nodes (counter) and the pointer (curPtr) to detect the end of the list.**

**a. Rewrite this loop to use only the pointer curPtr to detect the end the linked list (do not count the nodes).**

**int frequency = 0;  
   Node<ItemType>\* curPtr = headPtr;  
   while (curPtr != nullptr)  
   {  
      if (anEntry == curPtr->getItem())  
      {  
         frequency++;  
      } // end if  
        
      curPtr = curPtr->getNext();  
   } // end while**

**b. Rewrite this loop to use only the node count to detect the end of the loop. You will still need the pointer to traverse the list.**

**int frequency = 0;  
   int counter = 0;  
   Node<ItemType>\* curPtr = headPtr;  
   while (counter < itemCount)  
   {  
      if (anEntry == curPtr->getItem())  
      {  
         frequency++;  
      } // end if  
        
      counter++;  
      curPtr = curPtr->getNext();  
   } // end while**