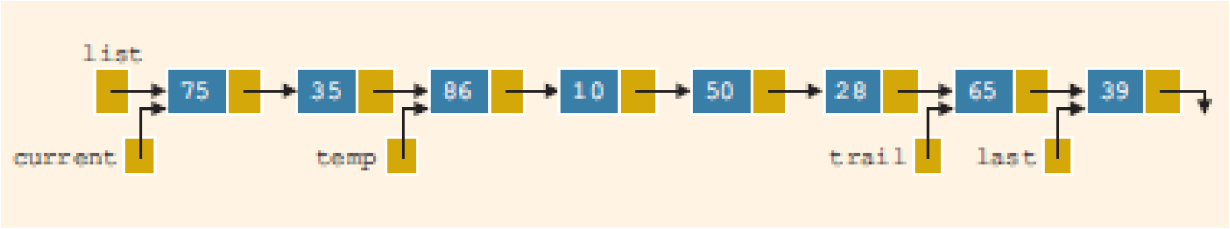
Name:Osbaldo Gonzalez Jr.

Linked List Worksheet

Consider the linked list shown below. Assume that the nodes are in the usual info-link form. Use this list to answer Exercises 1 through 6. If necessary, declare additional variables. (Assume that list, current, temp, trail, and last are pointers of type nodeType.)



1. What is the output, if any, of each of the following C++ statements?
   a. `cout << current->info;`
      75

   b. `current = current->link;`
      `cout << current->info;`
      35
   c. `cout << temp->link->link->info;`
      50
   d. `trail->link = NULL; //`**I'm assuming trail->link = NULL is only applicable for this question**
      `cout << trail->info;`
      65
   e. `cout << last->link->info;`
      last->link is NULL, therefore NULL->info is undefined
2. What is the value of each of the following relational expressions?
   a. `current->link == temp`
      false
   b. `temp->link->link->info == 50`
      true

   c. `trail->link->link == 0`
      true (NULL is equal 0)

   d. `last->link == NULL`
      true

   e. `list == current`
      true

3. What are the effects, if any, of each of the following C++ statements?

a. `trail->link = NULL;`

The linked list ends early. The node pointed by last is no longer part of the linked list. The new "last" should be the node pointed by trail. Assuming that the memory was being allocated dynamically, it is most likely that the node pointed by the "last" pointer will be lost, and a memory leak will occur.

a. `delete last;`

In order to reach the node pointed by "last", you would need to traverse the whole list. It would be a lot slower therefore. If any code depended on that "last" pointer, stuff would break. The linked list will not be modified though, so as a solitary list, it will operate more or less the same.

b. `temp->link = trail;`

The linked list would no longer include the nodes whose info attribute is 10, 50, and 28. A whole chunk of the linked list would therefore be bypassed.

c. `list->info = 19;`

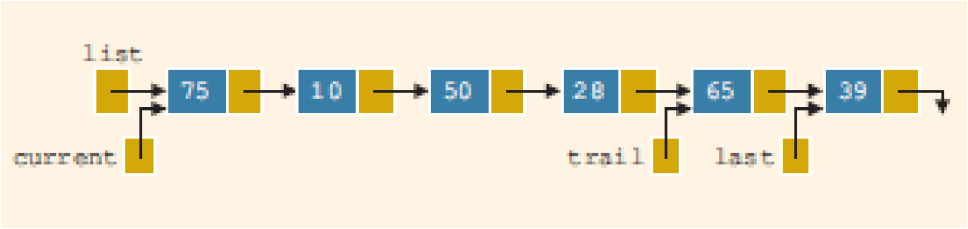
The value of the head node will change from 75 to 19. Nothing else will change.

d. `current = current->link;`

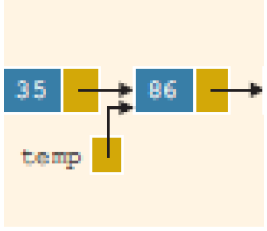
Current is now pointing to the node whose info attribute holds 35. Current basically moved one step down the list.

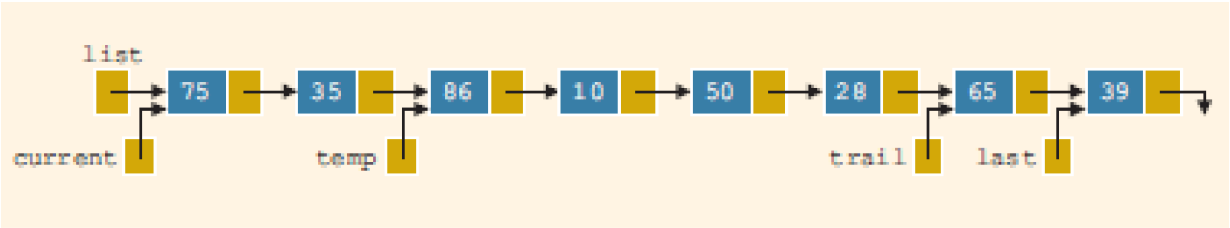
e. `current->link = temp->link;`

The node that the "current" pointer points to would now point to the node that the "temp" pointer's node points to… The structure would look something like this:



Note how these nodes are gone:





4. Write C++ statements to do the following:
   a. Set the info of the second node to 52.
      ```
      current->link->info = 52;
      ```
   b. Make current point to the node with info 10.
      ```
      current = current->link;
      ```
   c. Make trail point to the node before temp.
      ```
      trail = current->link;
      ```
   d. Make temp point to an empty list.
      ```
      temp = nullptr;
      ```
   e. Set the value of the node before trail to 36.
      ```
      temp->link->link->link->info = 36;
      ```
   f. Write a while loop to make current point to the node with info 10.
      ```
      while (current != NULL && current->info != 10)
            current = current->link;
      ```

5. Mark each of the following statements as valid or invalid. If a statement is invalid, explain why.

a. `current = list;` **//valid**

b. `temp->link->link = NULL;` **//valid**

c. `trail->link = 0;` **//valid**

d. `*temp = last;` **//invalid, dereferencing temp; assigning //nodeType to nodeType pointer.**

e. `list = 75;` **//invalid, list is type nodeType pointer, //75 is of type int: types don't agree //could do (nodeType *)75; but that's //probably dangerous**

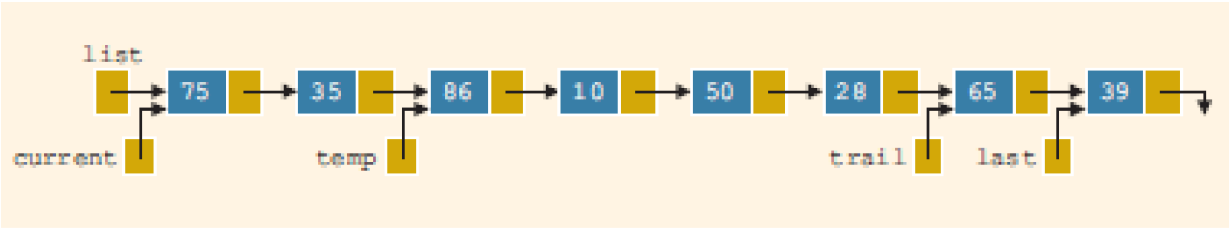
f. `temp->link->info = 75;` **//valid**

g. `current->info = trail->link;` **//Invalid, setting info which //is not a NodeType pointer //to a NodeType pointer will //result in an error**

h. `*list = *last;` **//valid**

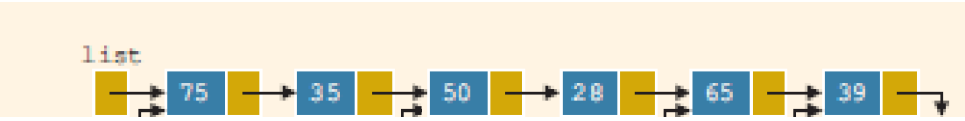
i. `current = last;` **//valid**

j. `cout << trail->link->link->info;` **//Invalid, even though //no compiling errors, //will be accessing NULL //pointer as if it is a //pointer to a NodeType //and therefore can have //undefined results.**



6. If the following C++ code is valid, show the output. If it is invalid, explain why.

```
current = temp->link;     //current = 10
trail = list;             //trail = 75
temp = list->link;        //temp = 35
trail = temp;             //trail = 35
temp->link = current->link; //35 -> 50, skip 86 and 10
```




```
current = trail->link;    //current = 50
cout << trail->info << " " << current->info << endl;
```

Output: 35 50

7. Show what is produced by the following C++ code. Assume the node is in the usual info-link form with the info of the type int. (list, trail, and current are pointers of type nodeType.)

```
list = new nodeType;
list->info = 28;
trail = new nodeType;
trail->info = 33;
trail->link = list;
list->link = NULL;
current = new nodeType;
current->info = 62;
trail->link = current;
current->link = list;
list = trail;
current = list->link;
trail = current->link;
cout << list->info << " " << current->info << " "
     << trail->info << endl;
```

Output: 33 62 28

8. Show what is produced by the following C++ code. Assume the node is in the usual info-link form with the info of the type int. (list, trail, and current are pointers of type nodeType.)

```
current = new nodeType;
current->info = 72;
```

```
current->link = NULL;
trail = current;
current = new nodeType;
current->info = 46;
current->link = trail;
list = current;
current = new nodeType;
current->info = 52;
list->link = current;
current->link = trail;
trail = current;
current = new nodeType;
current->info = 91;
current->link = trail->link;
trail->link = current;
current = list;
while (current!= NULL)
{
     cout << current->info << " ";
     current=current->next; //Assuming ->link was meant
}
```

Output: 46 52 91 72