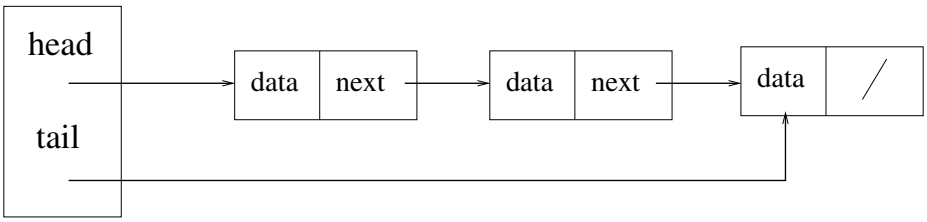
CSC112 Spring 2011
Fundamentals of Computer Science
Lab 13

In this week's lab, we will gain experience with both linked lists and inheritance in C++.

In our discussions of linked lists, we noted that the number of basic operations required to put a new element on the front of the list is independent of the length of the list. By contrast, putting a new element on the end of the list required traversing the entire list to find the end. With a very simple technique, we can make an improvement in the number of steps required to append an element at the end of the list. Instead of keeping only a `head` pointer, we keep both a `head` and a `tail` pointer. The idea is illustrated in the diagram below.



To get started, download the source code for a base class and main program which illustrates a linked list.

`http://menehune.opt.wfu.edu/Lab13/lab13_start.cc`

Your task: Complete the downloaded code to create a working program. To this end, create a derived class named `applist` using public inheritance from class `list`. You need to make the following changes to the downloaded source code:

1. Declare class `applist` to be a friend class of class `list_cell`. This is necessary because the derived class `applist` will need to access the `name` and `phone` data members of class `list_cell`.
2. Change the `private` designation in class `list` to `protected`. This is necessary to enable member functions in the derived class to access the `head` member in class `list`.
3. In class `applist`:
   (a) Add a data member named `tail` that points to the end of the list.
   (b) Add a default constructor for class `applist` that sets `tail` to `NULL`.

(c) Add a new member function `append`. Function `append` should accept two (const) character strings representing a name and a phone number. Use the `tail` pointer to **efficiently**[1] append a new name and phone number to the end of the list.

   i. Check for (and correctly handle) the special case when you are appending the a name and phone number on an empty list.

   ii. Be sure to update the list's `tail` pointer as you add each new element.

   iii. Function `append` should duplicate strings, similar to the function prepend.

(d) Add a new member function `unlisted` to class `applist`. Function `unlisted` should accept a character string representing a name, and it should change the corresponding phone number to "unlisted". Be sure to delete the old phone number provided it is not `empty`. Function `unlisted` should do nothing if the input name is not on the list.

(e) There is a prepend function in class `list` which is inherited by class `applist`. However, it does not work correctly with objects of class `applist`, because it does not update the `tail` pointer when prepending the first element on the list. Overload the prepend function in class `applist` to work correctly in that class.

Turn in: Change to the directory containing the sub-directory "Lab13". Create a file named "lab13.tar" using the command: `tar cf lab13.tar Lab13`. Upload the file "lab13.tar" to your account on telesto.

[1] The number of basic operations that your function performs should be independent of the length of the list.