CSE 12 Week Four, Lecture Two
(Today’s discussion: getting started on hw5)
Monday’s discussion: remove_List drawing, delete_List, Q&A

Hw5: MyRec struct:
- __________object that we can create to ___________.
- Representative of __________ to insert
- Structure with ______ data fields
- Calculator uses __________: ______ fields
- Provide constraint methods:
  o ____________
  o ____________
  o ____________
  o ____________
  o ____________
- Provide the constructor, also:
  o ____________

MyRec struct:
- Purpose: To be the test object to store, remove, view from
  your List (the code to be tested).
- Goal: To be simple, yet representative of ANY object that
  could be stored in the future.

MyRec methods:
- Purpose: To conform to the constraints imposed through
  the use of List.
  o To insert a MyRec object into the List, these
    functions must be provided.
- Constraints:
  o The functions need to be provided.
  o The parameters to those functions
    ________________:
      ■ ________________
      ■ ________________
      ■ ________________
- Bottom line:
  o ________________
- First lines of code in such functions:
  o ________________
  o ________________
Tiers or levels of code:
- Each tier related to surrounding tiers through calling methods in ________tiers, being called by methods of ________ tiers.
- Each method needs to work based on its _______ and ________ values.
- When all the tiers are put together, you have ________.

Outer Tier: User interface:
- Method of the ________ code in ________.
- Provides sample object used to test container:
  o ________ structure definition
  o ________ methods, expected by the ________ (constraints)
- Calls to methods of ________ (and ________ to test).
- Files: ________, ________, provided (except command line processing)

Next Tier: ________ code
- Methods of ________.
- Calls to methods of ________.
- Files: ________, ________, provided (except method comments).
- Methods called by “_______” code.

Next Tier: ________ code
- Methods of ________.
- Calls to methods of ________.
- File: ________, ________, most of your code to implement is here.
- Methods called by “_______” code.

Next Tier: ________ code
- Methods of ________.
- Calls to methods using ________ (_______ methods that exist back in ________and ________).
- File: ________, ________, some provided.
- Methods called by “_______” code.
Layering/Tiers of code:
Why:
- How OOD is done in industry.
- Small pieces can fit into your solution in only the correct way
  o Limits run-time errors.
  o It if compiles, it may just work.
- Encapsulating similar tasks with similar methods.
  o High level functions don’t care/know about the low level functions.
  o Low level functions don’t care/know about the high level functions.

Example: Writing the List to the user:

- Driver calls: _________________________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________:
  o ________is called if driver1 or driver2.

Example: Inserting into a List:

- Driver calls: _________________________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________:
  o ________is called if ________.

Example: Removing from a List:

- Driver calls: _________________________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________
- _______ _______ calls: _________:
  o ________is called if ________.
Note: driver1.c:
- No “________” is present
- No “________” is passed as ____________________
- driver1 does not know how to ________________
- Therefore, it can only ________________

The new_Node method:
What: The constructor for a Node.
When: Called every time you _________________________.
Responsibilities: ____________________________________.

Key line of code:
```c
this_node->data=(copy_func) ? (*copy_func) (element) : element;
```

(copy_func) ?
What: ____________________________________.
Purpose:
○ ____________________________________.
○ ____________________________________.

(*copy_func):
What: ____________________________________.
Purpose:
○ ____________________________________

(element):
What: ____________________________________
Purpose:
○ ____________________________________

: element:
What: ____________________________________
Purpose:
○ ____________________________________

Key point:
- ____________________________________?
- ____________________________________?
- ____________________________________?
- ____________________________________?
The delete_Node method:
What: The destructor for a Node.
When:
- It’s called ___________________________________.
  o ___________________________________.
- It’s called ___________________________________.
  *you could implement ______ by calling “_________
Responsibilities:
1. ___________________________________.
2. ___________________________________.
Key line of code:
if (delete_func && (*npp)->data)
  (*delete_func) (&((*npp)->data));

if (delete_func:
  What: _________________________________.
  Purpose:
  o ___________________________________.
  o ___________________________________.

&& (*npp)->data:
  What: _________________________________.
  Purpose:
  o ___________________________________.

*delete_func (:
  What: _________________________________.
  Purpose:
  o ___________________________________.

& ((*npp->data)):
  What: _________________________________.
  Purpose:
  o ___________________________________.
  o ___________________________________.
How can you call delete_Node to preserve the data?
1. ____________________________.
2. ____________________________.

Code differences in driver1.c and driver2.c:
   driver1: element is a _________
   driver2: element is a _________

   driver1: new_MyRec is __________, called by __________
   driver2: new_MyRec is __________, called by __________

   driver1: element points to __________ location with each insert call.
   driver2: address passed to insert is the __________
      - address of the __________ element.
      - list stores different items as it __________ of the current value of the “xxx” field of the MyRec parameter.

// a more readable delete_Node method: use a “this_Node” local pointer
static void delete_Node (Node ** npp, void (*delete_func) (void *)) {
    Node * this_Node;

    /* does the node exist?? */
    if (!npp || !*npp) {
        fprintf (stderr, DELETE_NONEXISTNODE);
        return;
    }

    this_Node = *npp;

    /* call function to delete element */
    if (delete_func && this_Node->data)
        (*delete_func)(&(this_Node->data));

    /* delete element */
    free (*npp);

    /* assign node to NULL */
    *npp = NULL;
}
Week Four, hw5 Discussion:
new_List constructor:
- Purpose: to ________ and to ________

```c
List * new_List (void *(*copy_func) (void *),
    void (*delete_func) (void *),
    FILE *(*write_func) (void *, FILE *)) {

}  
```

Insert: works at front, end, sorted
Remove: works at front, end
View: works at front, end

Approach to List’s Insert:
- Main goal: To locate the Node that you will insert before to achieve the desired insertion.
- Save prior value of “front” to be able to restore after the insertion is complete.
- Calling “advance_pre” or “advance_next” in a loop until reach that desired location (in find location)
- Once you find that location, you’ll call Node’s insert.
- Node’s insert will be the method that updates the “pre” and “next” pointers.
- Node’s insert will insert the item BEFORE the parameter Node.

```c
static Node * insert_Node (Node * this_Node, void * element, void *(*copy_func) (void *));
    Node * this_Node: The Node to insert after, NULL if initial insertion
    void * element: The data to store in a new Node
    copy_func: how to make a copy… 0 for driver1 (example follows below).
    return value: the Node inserted
```

For “driver1.c” (not driver2.c), calling new_Node in the following manner will work:
Node * working = new_Node (element, 0);
Some code in Node’s insert
- special case of inserting the first Node in the List.
  o if the Node * parameter is NULL, you know you are inserting the ____________.
  o method returns a pointer to the Node inserted.
  o List’s insert captures value and assigns to ________.
  o some code inside of insert_Node will be:
    ▪ this_Node->pre = …
    ▪ this_Node->next = …

One arrow in the drawing is one assignment in code.

To insert a Node before the parameter Node,
1. ________ the new Node to the list.
2. ________ the new Node into the list.
3. Back in List’s insert, check if inserting at the “front” and adjust __________________.