CSE 12 Week Five, Lecture One
Today’s discussion: hw5 detail & delete_List

Hash Tables:
What:
- A container object.

Behavior:
- An item is inserted into the table where ____________.
- An item is search for in the table where ____________.
- Often _______ based.
  o Fast access through use of ___________________.
- Often _____ sizes are ______.
  o _____ table size ensures that all locations are ________________.

Example:
- My keys:
  o Search in ________.
  o Search in ________.
  o Search in ________.

- Parking your car:
  o ________?

Index:  0  1  2  3  4

Where does an item go in the table?
To answer this question we need to ask the following question:
What is an aspect of a student is known regardless of the operation on the table (insert, lookup, remove).
Ans: _______ field: the known aspect.
For hw6:
- The _______ is the key field.
- The student number field will not be known for lookups.
  o Needing to know the student number is the motivation for the query!

Task: To translate the _______ into a numeric value (hash code/value/number) that can be used to locate the index into the array where object is to be stored.

How do we translate a key field into a numeric value?
One solution:
- ________________________.

Location in the table to store a student:
- ________________________
  - Example:

  __________________________________________
  __________________________________________

Increment: ________________________

To get the new location: _______________________________
Examples:

<table>
<thead>
<tr>
<th>Index:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.
2.
3.
4.
5.

Run the sample driver to see the table in action.

Let’s look at Lookup/Remove:

Tracker: sum: 716: Init: 1, Incr: 1, Sequence: 1 2 3 4 0

Ending a search:
1.
2.
3.

What to do about duplicate insertions into a container:

Ans: Look to _________________ to decide:

Choices:
- Disallow
- Count number of duplicates
- Replace the original.
C++;  // C language post incremented
Language feature of C++: Reference:
What: ____________ for something that exists elsewhere.
    - like an ________.

Implemented:
    - via ______ behind the scenes.
    - implementation is __________ to usage.
      o focus on a reference being just “__________”
    - _______ take the address of a reference:
      o references are only ________.
      o they don’t exist ______________________.

Syntax:
    - Ampersand in a declaration.
    - Same symbol as “address of”, yet used in the context of __________.
    - Establishing a reference must be done at __________ (mandatory).
    - References can _______be reset to refer to another variable within a function.
void func () {
    long lll = 0; // RTS variable
    long & lr = lll; // “lr” ______ to “lll”, syntax:
                     // reference establishment

    lr = 20; // ______________________
    long & lr2; // ______________________

    long * lp = &lll; // ______________________

    lp = &lr; // ______________________

    // above example is ______________
}

Real uses of references are:
- as ___________________
- as ___________________

Benefits of references:
- Get the power of ___________________
  o Same benefit as in Java.
- C++ extra benefits:
  o Use with ___________________
  o Compiler enforces ___________________.

When to use:
- ___________________!
Constructor and Destructor examples:

```cpp
void func () {

    UCSDStudent stu, stu2("Tracker"), * stup;
    // stu, stu2 ___________________
    // stu is initialized by ___________________
    // stu2 is initialized by ___________________
    // stup is ___________________:
        // like ____________________;      // in Java

    stup = new UCSDStudent;
    // calls __________________for a _______ object

    stup = new UCSDStudent ();      // __________

    stup->number = 10; // accessing field via pointer
    stu.number = 10; // accessing field via an object

    delete stup;      // call to __________________
} // end of scope…call to __________________
// on ___________________________

<discussion of differences of where to allocate objects>
```

Optional parameters:
What: any number of trailing parameters passed to a function that provides defaults values for parameters not found when calling function.
Where: Can be any function…not just constructors.
Guaranteed initialization:
What: special syntax for constructors.
Why: to initialize data fields
Syntax: data fields are listed in a comma separated list in the order declared with each initial value listed (in parenthesis) after a colon before open curly of the constructor.
Example:

Constructor (void) : datafield (init_value), datafield2 (initial_value), datafield3 (init_value) { …}

// normally: in UCSDStudent.h:
class UCSDStudent {   // first section is private (default)
    char name [20];  // private field
    long number;  // private field

public:
    UCSDStudent (void) {}  // __________
    UCSDStudent (char * nm, long nmbr = 0) :
        number (nmbr) {
            strcpy (name, nm);  // G.I. can’t initialize arrays
        }
    ~UCSDStudent (void) {}  // __________

    void setNumber (UCSDStudent &);  // __________
};  // semi-colon is ______________

Scope Resolution:
Scope: where a symbol is __________
Resolution: to resolve ___________
Syntax:
C++: ClassName :: symbol_to_resolve
Java: ClassName.symbol_to_resolve
Uses:
C++/Java: __________________________
C++: __________________________
Member methods defined outside class definition:

// normally: in UCSDStudent.c:  
void UCSDStudent::setNumber (UCSDStudent & stur) {  
    number = stur.number; //this->number = stur.number  
    stur.number = 0;  
}

main () {  
    
    UCSDStudent stu ("Tracker", 123), stu2;  
    stu2.setNumber (stu); // pass by __________  
    // stu’s number = 0…changed __________  
}

Think about for future:  
1. Why define a method outside/inside its class definition?  
2. Why/when to use guaranteed initialization syntax?
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:
```
this_node->data=(copy_func) ? (*copy_func) (element) : element;
```

(copy_func) ?
What: ____________________________
Purpose:
  o ________________________________.

(*copy_func):
What: ________________________________.
Purpose:
  o ________________________________

(element):
What: ________________________________.
Purpose:
  o ________________________________

: element:
What: ________________________________.
Purpose:
  o ________________________________.

Key point:
- ____________________________________?
- ____________________________________?
- ____________________________________?
- ____________________________________?
The delete_Node method:
What: The destructor for a Node.
When:
- It’s called ___________________________________.
  o ___________________________________.
- It’s called ___________________________________*
  o ___________________________________.
*you could implement __________ by calling “__________”

Responsibilities:
1. ___________________________________.
2. ___________________________________.

Key line of code:
```c
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 ___________________________________.
  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;
}
The delete_List method:
Choices on implementation:
  1. ____________________________
  2. ____________________________

Points to consider:
  If take approach 2, you need to call ____________
  - a direct call to free is insufficient from delete_List.
    o Why: ______________________________

  If take approach 1, what do you have to keep in mind?
  ______________________________________

Advice to make code more readable in delete_List:
List * this_List;
...
  this_List = *lpp;

Now you can use this_List like all other methods in most of the code.

What if driver1 works, driver2 works and calc doesn’t?

Printing a Node using the debugger:
  (gdb) print *this_list
$1 = {end = 0x24178, list_count = 1, occupancy = 3, copy_func = 0,
    delete_func = 0x108f4 <delete_MyRec>, write_func = 0x10924 <write_MyRec>}
  (gdb) print *(this_list->end)
$2 = {pre = 0x24148, next = 0x24160, data = 0x23d60}
  (gdb) print *(this_list->end->next)