Constraints:
What are the constraints on an object to be a candidate for insertion into the generic container using C++’s templates?
Ans:

- Provide:

- Compiler will ______ when constraints are not met.

Guaranteed initialization:
- Constructor syntax used to initialize data fields when instantiating an object.
- Sometimes optional:
  - ______________
  - ______________
- Sometimes mandatory:
  - ______________
  - ______________
  - ______________
  - ______________
  - ______________
- Sometimes not possible:
  - ______________
Member methods:
- Must be declared in the class definition.
- Can be defined outside the class definition.
- Reasons to declare method outside class definition:
  o Proprietary algorithms.
  o Lengthy functions.
    ▪ No Javadoc standards.
    ▪ Ideally…a class definition in C++ is small
      • Want to see class in about a screen full.

Inline expansion:
- A request to the compiler to expand a method body into the code at the location of a method call.
- Similar to the #define INDEX, CHARACTER, PRIORITY preprocessor directives of hw2.
- Syntax:
  o “inline” keyword prefix to functions not in a class definition.
  o Automatically request for member methods defined within class definition.
- When:
  o For small-bodied functions called frequently.
  o For member methods, for methods where company secrets are not revealed.
- Request denied for methods with loops.
```plaintext
int main () {
    Parent MomDad;       // ___ bytes
    Private Eye;         // ___ bytes
    Protected Wilderness; // ___ bytes
    Public Property;     // ___ bytes

    // scope resolution
    main  Parent::
    MomDad.aaa = 10;
    MomDad.bbb = 10;
    MomDad.ccc = 10;
    Eye.aaa = 10;
    Eye.bbb = 10;
    Eye.ccc = 10;
    Eye.ddd = 10;
    Eye.eee = 10;
    Eye.fff = 10;
    Wilderness.aaa = 10;
    Wilderness.bbb = 10;
    Wilderness.ccc = 10;
    Wilderness.ddd = 10;
}
```
<table>
<thead>
<tr>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wilderness.eee = 10;</td>
</tr>
<tr>
<td>Wilderness.fff = 10;</td>
</tr>
<tr>
<td>Wilderness.ggg = 10;</td>
</tr>
<tr>
<td>Wilderness.hhh = 10;</td>
</tr>
<tr>
<td>Wilderness.iii = 10</td>
</tr>
<tr>
<td>Property.aaa = 10;</td>
</tr>
<tr>
<td>Property.bbb = 10;</td>
</tr>
<tr>
<td>Property.ccc = 10;</td>
</tr>
<tr>
<td>Property.jjj = 10;</td>
</tr>
<tr>
<td>Property.kkk = 10;</td>
</tr>
<tr>
<td>Property.iii = 10;</td>
</tr>
<tr>
<td>aaa = 10;</td>
</tr>
<tr>
<td>bbb = 10;</td>
</tr>
<tr>
<td>ccc = 10;</td>
</tr>
<tr>
<td>ddd = 10;</td>
</tr>
<tr>
<td>eee = 10;</td>
</tr>
<tr>
<td>fff = 10;</td>
</tr>
<tr>
<td>ggg = 10;</td>
</tr>
<tr>
<td>hhh = 10;</td>
</tr>
<tr>
<td>iii = 10;</td>
</tr>
<tr>
<td>jjj = 10;</td>
</tr>
<tr>
<td>kkk = 10;</td>
</tr>
<tr>
<td>iii = 10;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Iterator: A way to sequentially access (“traverse”) each object in a container or collection
  - Access is done as needed, not necessarily at once.
  - Implemented by a private pointer advancing to the next item or prior item as needed.
    - Implementation differs based on container
      - class List {
          ▪ Node * end;
          ▪ Node * iterator;
        }
  - Operations (customize as needed):
    - getCurrentItem – returns item at iterator
      - return iterator->data;
    - hasNext – returns true if there is a next item
    - getNextItem – iterates to the next item
      - iterator = iterator->next;
      - return getCurrentItem();
    - hasPrior – returns true if there is a prior item
    - getPriorItem – iterates to the prior item
      - iterator = iterator->pre;
      - return getCurrentItem();
    - reset – resets private pointer to first item.
      - iterator = iterator->end;
Discussion:

Templates:
What: The C++ language feature that implement polymorphic generic containers.
Similar to Java’s “generics”
Many benefits: Try to find them. We’ll address them together later.
Syntax: One part:
  - Instantiating the container:
    o Tell the compiler the type of object that you will be storing.
      ▪ Pass instantiation argument in between <>
        • Ex: SymTab<UCSDStudent> ST;
      ▪ Almost like a parameter to the compiler.
        • At compile time, the compiler substitutes your instantiation argument for all uses of the template throughout the template code.

Syntax: Other part:
  - Declaring your generic container object:
    o Add a prefix to all blocks of code that use the template object:
      ▪ “template <class Whatever>”
  - Use Whatever as the name of the type of the generic object throughout the container code.
  - Type of container is: Type<Whatever>
    o Tree, TNode are not types anymore.
    o Tree<Whatever>, TNode<Whatever> are the types of the objects in code.
    o Tree<UCSDStudent>, TNode<UCSDStudent> are the types of the objects in the debugger.
To insert a break point at Tree’s Insert in gdb:
    break Tree<UCSDStudent>::Insert

Use of a reference parameter without a const prefix means that the method will ____________________________!

Occupancy will be a reference field of TNode:
    - refers to the occupancy in Tree
    - occupancy++;  // Tree code or TNode code
        o increment the occupancy in the Tree to which the TNode belongs!

How to start:
    - “make directories, make install”
    - UCSDStudent
        o Similar to Variable.
    - Copy bodies of your Insert method from your Tree.c code of hw7 into the bodies of Tree.c Insert code for hw8.
        o Split “Insert” between Tree and TNode methods.
        o Refer to recursive Tree Insert notes for help in splitting loop based Insert into a recursive Insert.
    - Complete other methods with null bodies:
        o {}  
        o {return 0;}
    - Compile
    - Test
    - SetHeightAndBalance w/o threshold on balance
    - Verify Insert works, and then go to Lookup.
    - Remove for 0 children and 1 child case
    - Remove for 2 children case
        o RARM
    - SetHeightAndBalance w/ threshold on balance
ReplaceAndRemoveMax is only called from ________.
- Only when Remove sees the item to remove is in TNode with two children.

How to compare the data in the elementTNode with the data in the current TNode:
// code within TNode’s Remove
if (elementTNode.data == data) {
    // calls operator == on your data to compare names

Inside Tree’s Insert:
    root->Insert (...); // calls TNode’s Insert
    … use arrow with pointers

TNode’s Remove, parameter “fromSHB = FALSE”
- Tracks if calling Remove from SetHeightAndBalance or not:
    o If yes…don’t again call SetHeightAndBalance from Remove.
        ▪ SetHeightAndBalance on that TNode will happen when it is ________________.
        • ________ always calls SetHeightAndBalance

Why no parent pointer in hw8?
Ans: Parent pointer was used to ________ in hw7. This functionality is handled through ________ in hw8.

Why pass a parentTNode to TNode constructor?
Why pass a Tree to the TNode constructor for root TNode?
Ans: To be able to ______________________________:
    ____________________
The reference data fields of TNode are __________________________.

ReplaceAndRemoveMax:
How do you replace the data in the target TNode with the data from the immediate predecessor?
Answer: _____________________!!!
- __________________________________________________________
- __________________________________________________________.
- Once you find the immediate predecessor, the data in the _____________ can be reassigned.
  o The _____________ was ________________ as you’ve moved down the Tree in your RARM calls.

Remove: elementTNode parameter:
- What: The Reference TNode that is a container that will hold ______________________.
- What is special about it?
  o Origin:__________________________.
  o Location in memory:______________.
  o Anything else:__________________.
- How: Passed ______________________.
- Initial contents: ___________________.
  o name ________, number ________.
- Final contents: ___________ UCSDStudent
  o name ________, number ________.
  o assigned by ____________________.
- Remember: send back _____ from elementTNode before returning from Tree’s Remove.
- Could we have used a Whatever & instead?
  o ______________________________.
Use of a reference parameter without a const prefix means that the method will change the parameter object!

TNode elementTNode parameter to TNode’s Remove is special:
- The only TNode that exists in your program that
  
  - Exists on ______:
    - Why: _______________________.
  - When does it originate:
    - ________________
- You can create by calling _____ TNode constructor:
  - Constructor where Parameter is a TNode:
    - What is its parent (needed by TNode constructor)?
      - ________________.
  - Constructor where Parameter is a Tree:
    - What is the “theTree” parameter?
      - ________________.
  - What about occupancy?
    - Occupancy is _________ in the TNode constructor.
  - Won’t this RTS TNode that isn’t part of the Tree skew the occupancy of Tree?
    - Yes. While Remove is executing, occupancy will be ______, but 
      ________________.
    - No. Because ________________ ________________.
Hard letters in the alphabet to insert:
  - F
  - G
  - P

F & G:
  Problem resolution: Look at the calls to Remove and Insert of the item removed within the body of SetHeightAndBalance.

P: Problem resolution: Look at your calls involving PointerInParent:
  Ex: left->Function(…, right); // inconsistency