**DATA STRUCTURES CRIB SHEET – GABE MAAYAN**

**CLASSES:**

public – everything has access, private – only member functs have access, protected – inheriting classes have access

If there is dynamic memory use, must have default and copy constructors (Office(), Office(const Office& o)), assignment operator (Office& operator=(const Office& o)), destructor(~Office())

In .h, int getNum(); in .cpp, int Office::getNum() {}

Structs are classes where everything is public

Templated classes – template <class T> class Test {}; template <class T> int Test<T>::getNum() {}

typedef map<pair<bool, int>, vector<string> > MY\_MAP;

**STRUCTURES+POINTERS:**

int\* p; \*p=10; --- int x=5; p=&x; \*p==5; x=8; \*p==8; --- int n=10; int\* a = new int(n); a is beginning of array; delete[] a; --- int\*\* a2 = new int\*[rows]; for each row, a[i]=new int[cols];

**VECTOR:** no find(), void pop\_back(), iterator insert(iterator pos, value), iterator erase(iterator pos), easy access and can use [], inserting and erasing is slower because of occasionally resizing

**DSLIST:** two way nodes, Node<t>\* head\_ and tail\_; unsigned int size\_;

**LIST:** can’t use [] to access, inserting and erasing is very quick

**TREES:** binary tree – can have 2 children, binary search tree – all nodes to the left of root are <root, all to right are >root, balanced tree – the # of nodes in each subtree is approx. =

**MAPS:** map<key, value> name;, entries are pairs, keys are constant, search, insert, erase are all O(logn), must have operator< defined for key, no duplicate keys, can use [] – if key isn’t there, inserts key and initializes value to default otherwise returns the value for the key, iterator find(key) – returns iterator of pair<key, value> or map.end()

**PAIRS:** pair<type1, type2>, pair<int, string> p = make\_pair(2, “test”), p.first==2, p.second==”test”

**SETS:** like maps but just keys, keys are constant – must remove then change and reinsert, pair<iterator, bool> insert(key) – iterator is location-bool is if inserted, erase(key)-returns # of entries remove, or erase(iterator), const\_iterator find(key)

**DSSET:** uses TreeNodes, insert always puts in null node, erase(if not a leaf or only 1 child) find node with greatest value in left or smallest in right and put that in the erased position

**HASH TABLES:** array/vector at top level, not a sorted order, each element is mapped to a slot with a hash function, hash function is a function that takes in a key and returns an int that maps to a spot in an array, elements are distributed uniformly throughout array, if more than one element is mapped to the same spot – collision, SEPARATE CHAINING – each spot is a linked list, works when number of items in each list is small, OPEN ADDRESSING – move to a different spot either with linear/quadratic probing or a secondary hashing function

**FUNCTOR:** Function as its own class – template <class T> class less { public: bool operator() (const T& x, const T& y) const {return x<y;}

**QUEUES:** First in first out, can only insert at back and remove from front – mostly uses lists

**STACKS:** First in last out, can only insert and remove from back – use vectors

**PRIORITY QUEUE:** think like a tree with top item has lowest priority, main operations insert, push, and pop(delete min), made out of a **binary heap** – binary tree s.t. each internal node is < both children, drawn as trees, but implemented WITH VECTORS (PARENT IS AT [(i-1)/2], LEFT IS AT [2i+1], RIGHT IS AT [2i+2]), pop replaces root with last leaf and then percolates down, push makes new last leaf then percolates up

**TRAVERSALS AND SEARCHES:**

In-order – left, root, right

Pre-order – root, left, right --- used to create copy of tree

Post-order – left, right, root --- used to delete tree

All of the above are depth first, this can take a long time if makes a wrong decision early

Breadth first visits node based on distance from the root – very memory intensive

**OPERATORS:**

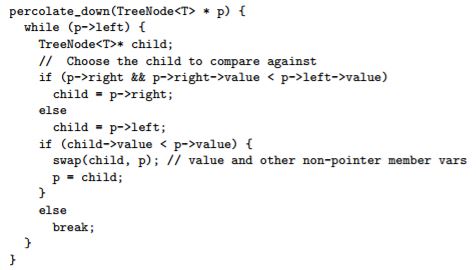
Office& operator= (const Office& old); - returns \*this Office operator+ (const Office& rhs) const; Office operator- () const; - negates, or Office operator-(const Office& rhs) const;

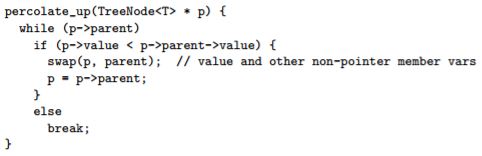
friend istream& operator>> (istream& istr, Office& o); -returns istr

ostream& operator<< (ostream& ostr, const Office& o); - non-member funct

Overload in this order -> Non-member, member, friend

**USEFUL FUNCTIONS:**



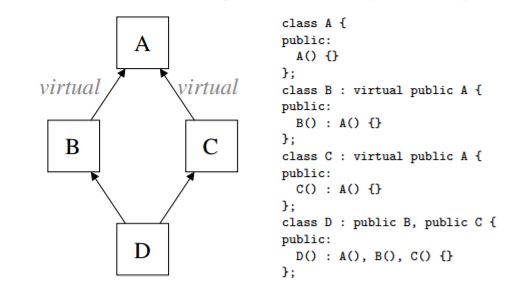
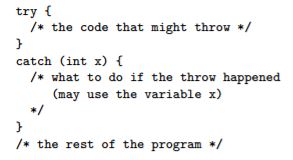


**INHERITANCE/EXCEPTIONS/GARBAGE/CONCURRENCY:**

Derived classes call base class constructor immediately, derived destructors go first then base dest.

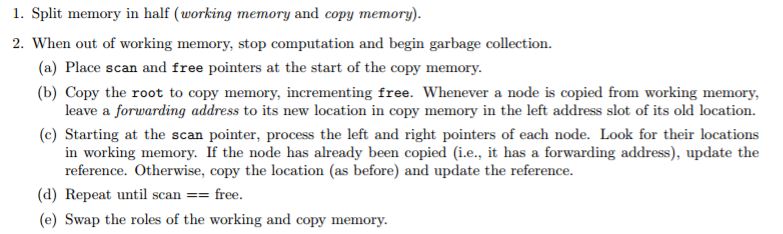
class Office : public Building {} – public means no status change, protected makes public protected, private makes everything private

if two go to the same base class and then another class inherits from the two, the two need to inherit base virtually



**REFERENCE COUNTING:** Attach a counter to each Node, Increment counter whenever a new pointer connects to the node, when pointer is removed, decrement counter, when counter==0, node is garbage and can be reused/deleted

**STOP + COPY:**



**MARK-SWEEP:**

