ECE264: Advanced C Programming

Summer 2019

Week 6: Exam2 Review, Priority Queues, Trees, Binary Trees

Segmentation Faults

- Cause 1: Invalid memory access
  - Accessing memory at address 0 (i.e. dereferencing a NULL pointer)
  - Accessing memory out of scope (dereferencing address of a variable outside where it is defined)
  - Accessing memory that we no longer own (i.e. dereferencing pointer that is freed)
  - Accessing uninitialized pointer
- Cause 2: Using up all the memory
  - Example: stack overflow

Example: using memory after **`free`**

- Releasing memory allocated to a linked-list

```
void DeallocateList(Node* head) {
      Node* temp;
      for(temp=head;temp!=NULL;temp=temp->next) {
           free(temp);
      }
}
```

Example: using up all memory

- Factorial

$$n! = \begin{cases} \texttt{n x (n-1)!} & \texttt{when n>=1} \\ \texttt{1} & \texttt{when n=0 // factorial of negative numbers not defined.} \end{cases}$$

```
int factorial(int n) {
    if(n == 0)
        return 1;
    else
        return n * factorial(n-1);
}
```

Segmentation Faults – Preventive Measures

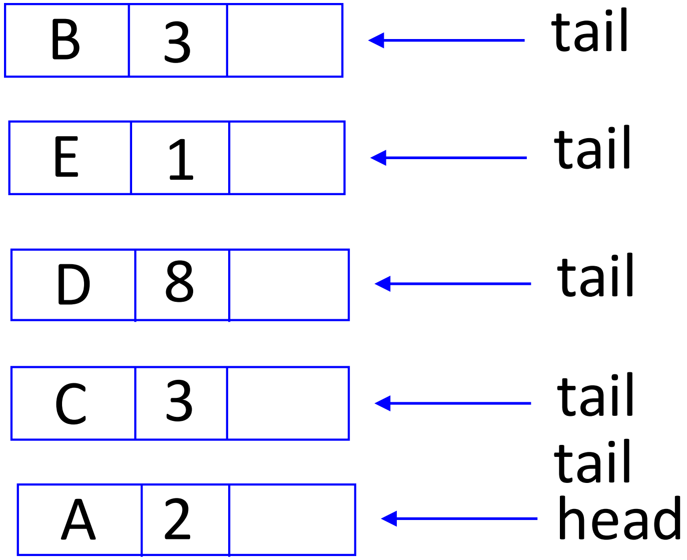
- If a pointer is returned, always check the return value for NULL
- Always initialize pointers to NULL
- Set pointers to NULL after freeing

Priority Queues (brief intro)

- Special types of queues: every item in the queue has a priority associated with it
- Enqueuing is same (as in normal queues)
- Dequeuing is different:
  - item with higher priority is dequeued before one with lower priority
  - If two items have same priority, the item that is ahead (closer to head) in queue is dequeued first

Priority Queues (insertion)

```
PriorityQueue pq={.head = NULL, .tail=NULL};
Enqueue(&q, <A,2>)    //<X,Y>: X is data, Y is priority, larger Y indicates higher priority
Enqueue(&q, <C,3>)
Enqueue(&q, <D,8>)
Enqueue(&q, <E,1>)
Enqueue(&q, <B,3>)
```

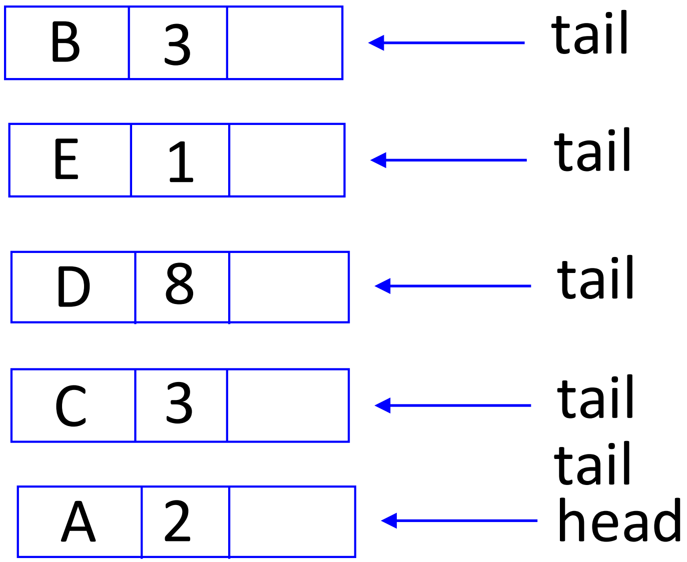


Priority Queues (deletion)

```
PriorityQueue pq={.head = NULL, .tail=NULL};
Enqueue(&q, <A,2>)  //<X,Y>: X is data, Y is priority, larger Y
                    indicates higher priority
Enqueue(&q, <C,3>)
Enqueue(&q, <D,8>)
Enqueue(&q, <E,1>)
Enqueue(&q, <B,3>)

retval=Dequeue(&q) //gets
highest priority element = D

retval=Dequeue(&q) //returns C,
since B is ahead in queue order
```



- Applications:
  - CPU assignment to processes
  - Computing shortest paths

Represented as trees

Trees

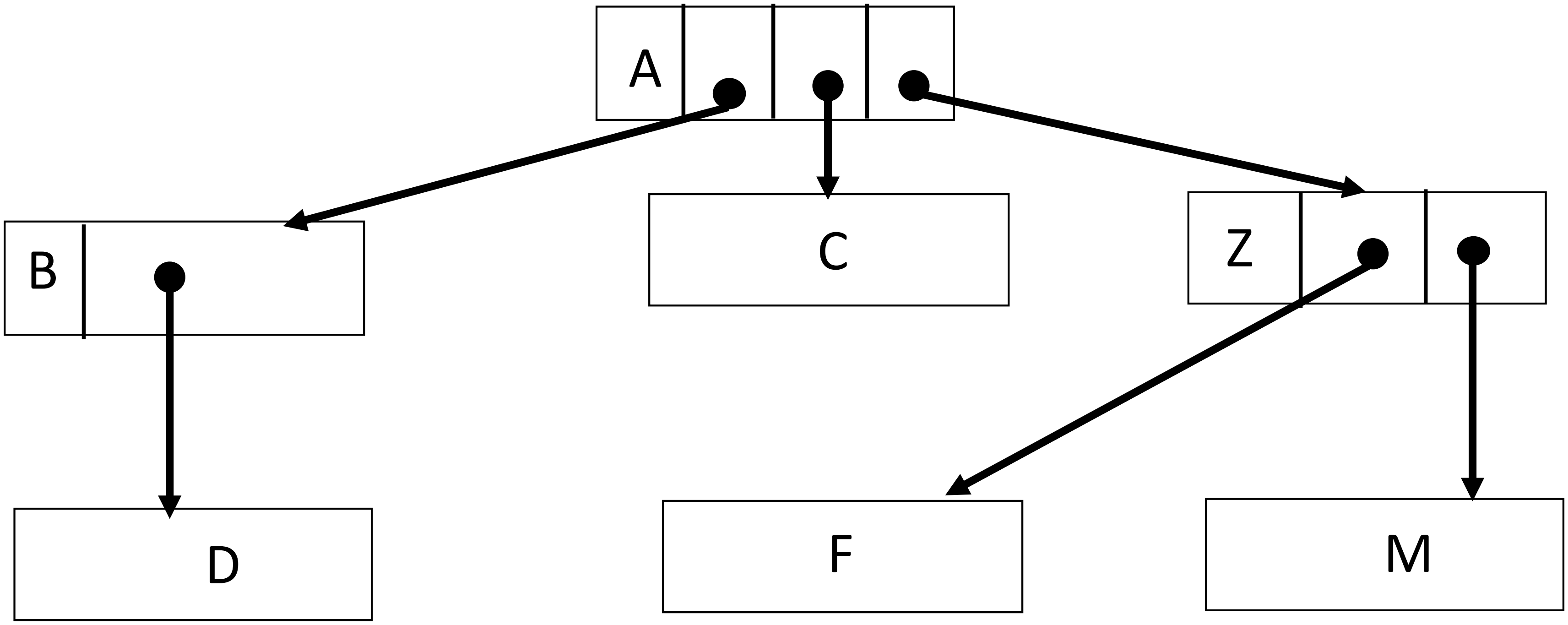
- Linked Lists, Stacks, Queues are linear data structures
  - One item follows another
- Trees are non-linear data structures (also called as hierarchical data structures)
  - More than one item can follow an item
  - The number of items that follow can vary from item to item

Trees

- Uses:
  - Organizing files in a disk
  - Simulating galaxies
  - Suggesting items bought together in a web shopping (e-commerce) portal

Trees - representation

- As a set of nodes connected on a plane:



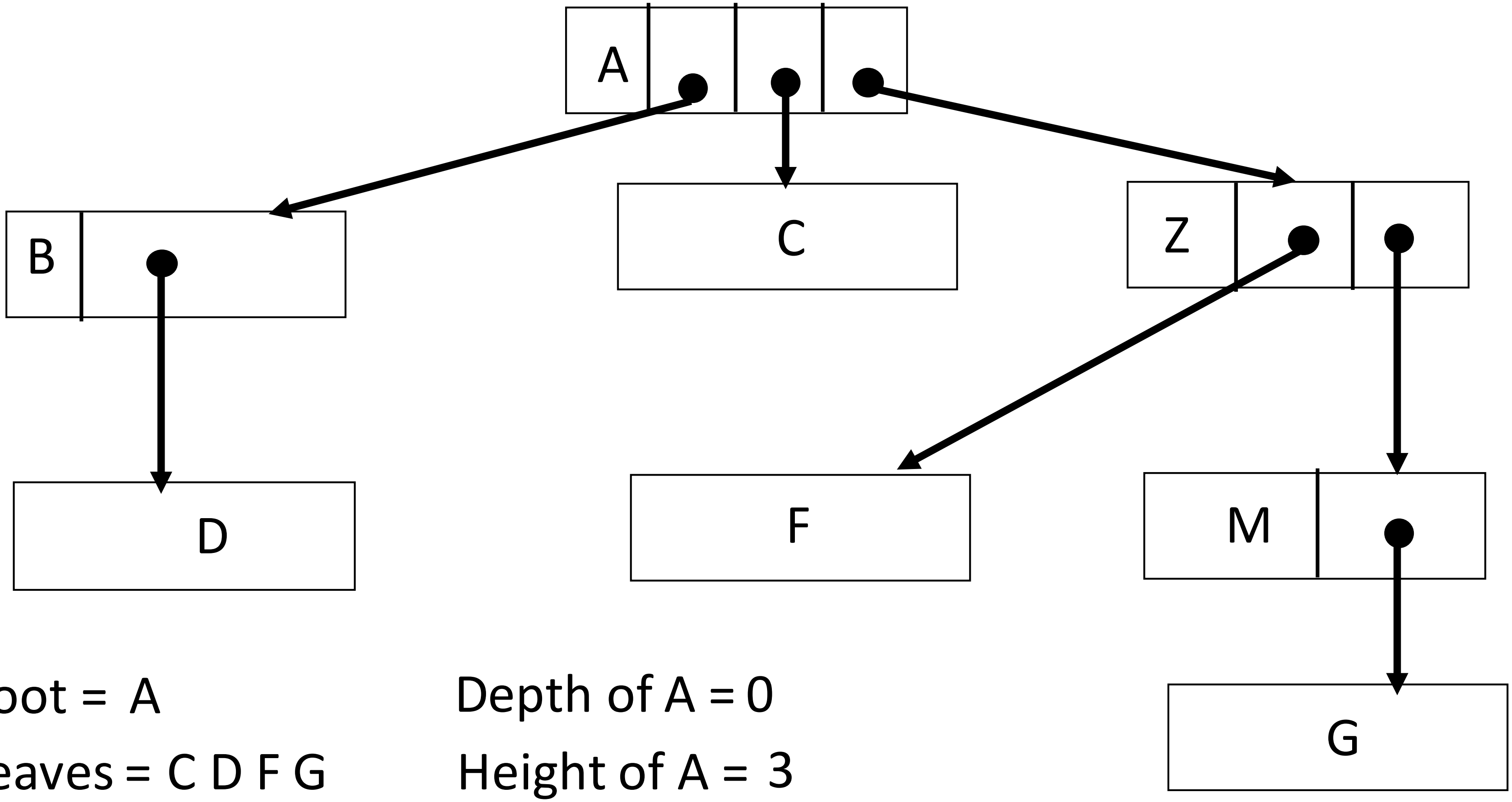
Trees - terminology

- Elements of a tree: nodes and edges
  - A node holds data and connections (references) to other nodes
  - An edge connects two nodes
- Every node connected by an edge from exactly one node (parent)
- Each node can be connected to any number of nodes (children)

More terminology

- **Root:** node at the top
- **Leaves:** bottom most nodes
- **Depth of a node (level):** number of edges from root to the node
- **Path in a tree:** sequence of zero or more connected nodes. **Path length** is the number of edges in the path* (Alternative definitions exist).
- **Height of a node:** number of edges from the node to the deepest leaf

Exercise



Root = A

Leaves = C D F G

Depth of F = 2

Depth of A = 0

Height of A = 3

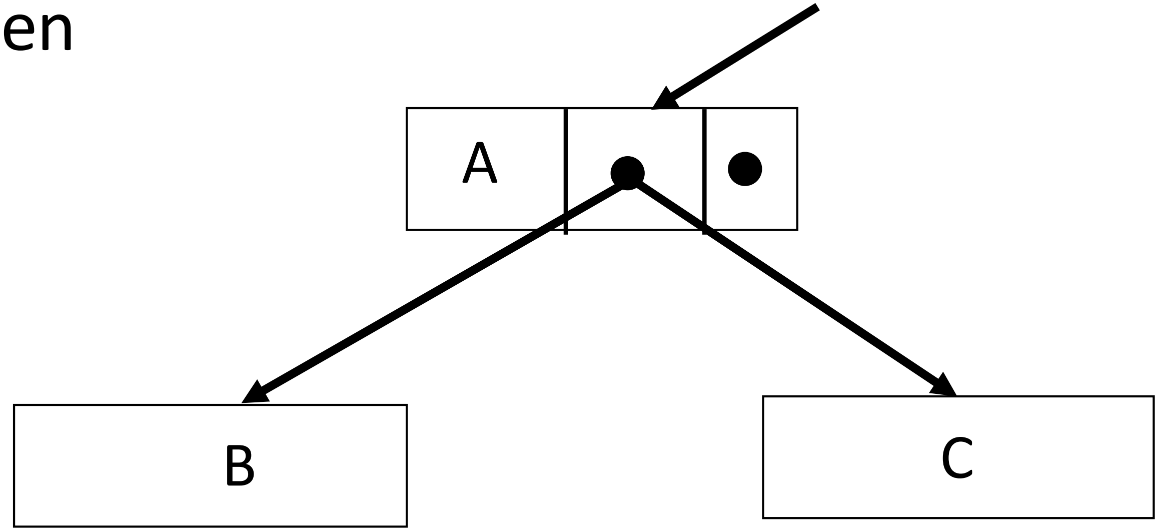
Height of Z = 2

Path length from D to G = ?

Path length from Z to G = 2

Binary Trees

- Trees with at most two children per node
- Given



- A is parent, B and C are children, B is left child, C is right child.

- **Subtree of a node:** includes one of node’s children and all of its *descendants*
- **Descendants of a node:** all nodes reachable from that node

Binary tree traversals

- Process of visiting *all* nodes in the tree
- Why?
  - To print all values
  - To check nodes with interesting properties
- Order
  - Breadth-first
  - Depth-first
    - Preorder, inorder, postorder