CSE 374: Programming Concepts and Tools

Spring 2026
Instructor: Megan Hazen

Today's Goals

Subject Matter

- Linked Lists
- Multiple files

Your Goals

- Look at info for Assessment 3
- Get started on HW4

Struct Tags: typdef - inline

- Has type **`struct person_info`**
- ‘Person_info’ is a *struct tag*, not a *type*
- Can use typedef to rename

```
typdef struct person_info {
  char * name;
  int age;
} person_info;

person_info *me;
person_info you;
me->age = 99;
you.age = 25;
```

Data Structures

- Organization of data that has some internal structure
  - Relationship is encoded in the way the data is stored
  - Dictates the types of operations that are performed on the data
- Arrays (very simple – just adjacent values)
- Records (associated values – essentially a struct)
- Hash tables (allow look up by value)
- Graphs
- Lists: stacks / queues
- Trees

Data Structures in C

- Organization of data that has some internal structure
  - Relationship is encoded in the way the data is stored
  - Dictates the types of operations that are performed on the data
- Arrays (very simple – just adjacent values)
- Records (associated values – essentially a struct)
- Hash tables (allow look up by value)
- Graphs
- Lists: stacks / queues
- Trees

- We build records / nodes
  - Define a struct to contain values
- We use pointers to associate nodes with other nodes
- C is procedural, not objected oriented
  - We write functions that act on nodes
  - Instead of nodes having methods that act on themselves….

Lists

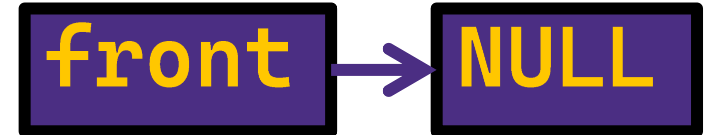
```c
// A single list node that stores an integer as data.
typedef struct IntListNode {
  int data;
  struct IntListNode* next;
} IntListNode;


// Allocates a new node on the heap.
IntListNode* makeNode(int data, IntListNode* next);
```

List nodes - making

```
IntListNode* makeNode(int data, IntListNode* next) {
  IntListNode* n = (IntListNode*) malloc(sizeof(IntListNode));
  if (n) {  // malloc might return null
    n->data = data;
    n->next = next;
  }
  return n;
}
```

Lists - building



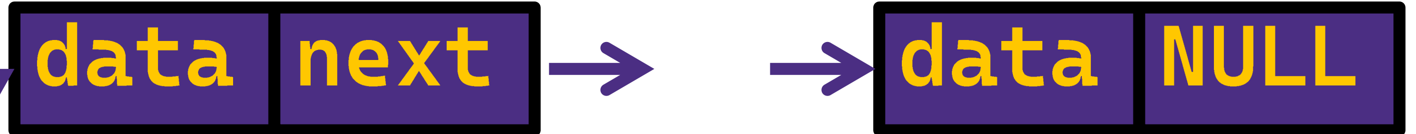

```c
IntListNode* fromArray(int* array, int length) {
  IntListNode* front = NULL;
  for (int i = length - 1; i >= 0; i--) {
    front = makeNode(array[i], front);
    if (front == NULL) {
      return NULL;
    }
  }
  return front;
}
```

List nodes – building, one node

```
IntListNode* fromArray(int* array, int length) {
  IntListNode* front = NULL;
  for (int i = length - 1; i >= 0; i--) {
    front = makeNode(array[i], front);
    if (front == NULL) {
      return NULL;
    }
  }
  return front;
}
```

Data structures – keeping track

```
List {
   node* Head;
   node* Tail;
}
```



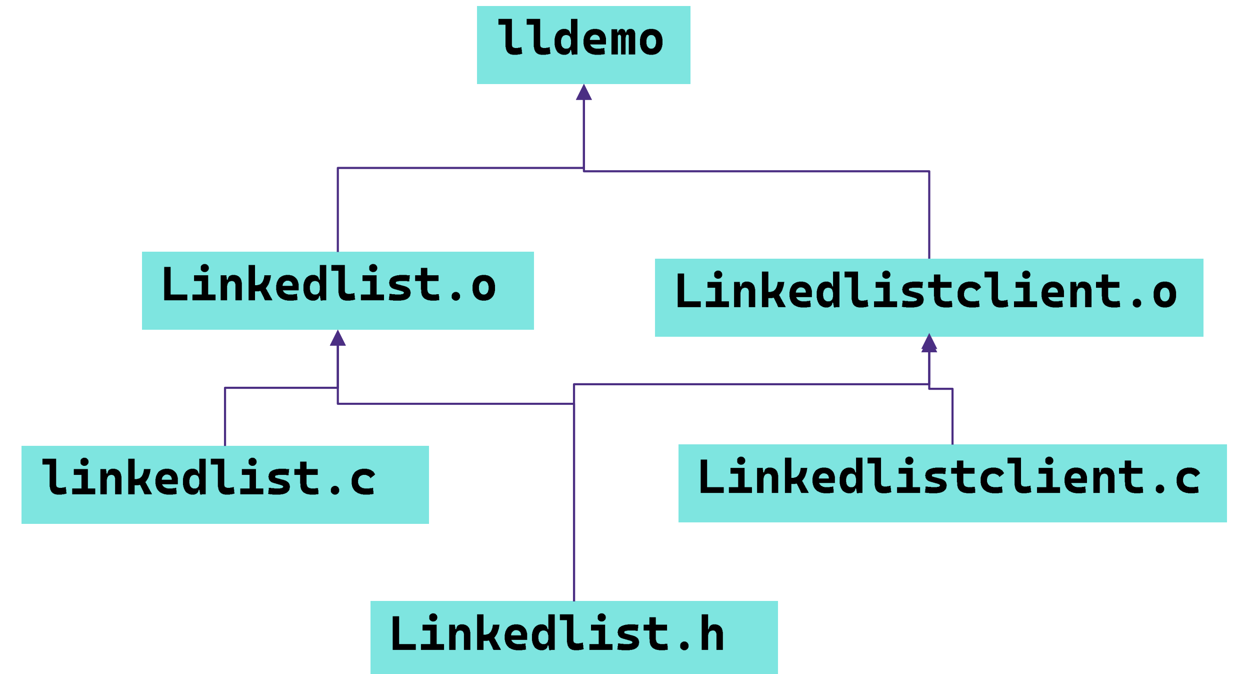
- It is important not to lose your pointers
  - Create another data structure that always holds these important addresses?

Data structures - freeing

```c
void freeList(IntListNode* front) {
  while (front != NULL) {
    IntListNode* next = front->next;
    free(front);
    front = next;
  }
}
```

Building library modules

- One set of code to define linked list:
  - Linkedlist.h
  - Linkedlist.c
- Another piece of code uses it:
  - Linkedlistclient.c
  - #include "linkedlist.h"



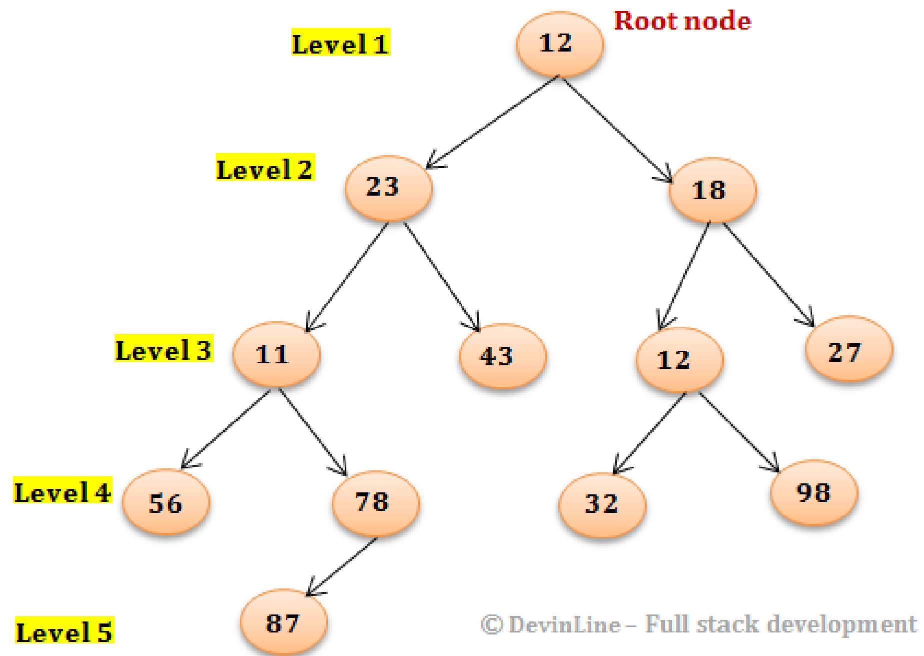
- Compile with

```
$gcc -o lldemo linkedlist.c linkedlistclient.c
```

Challenge

- What code goes in linkedlist.h?
- What code goes in linkedlist.c?
  - What are the includes?
- What code goes in linkedlistclient.c?
  - What are the includes?

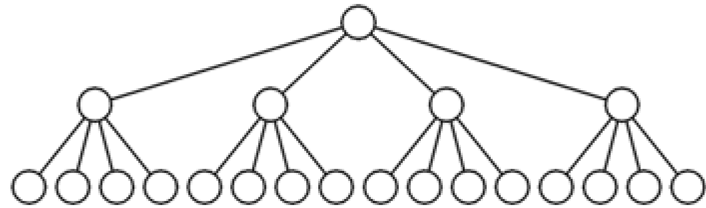
Binary Trees




```
struct BinaryTreeNode {
  int data;
  struct BinaryTreeNode* left;
  struct BinaryTreeNode* right;
}


struct BinaryTree {
  Struct BinaryTreeNode* root;
}
```

N-ary Trees

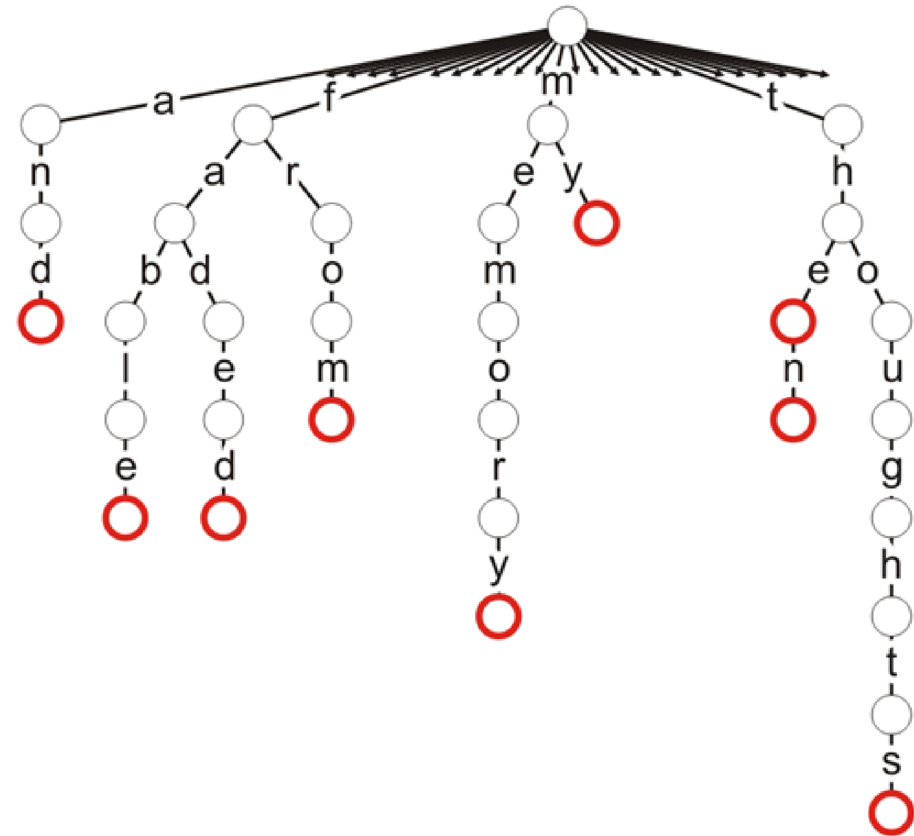


- Arbitray branching-number
- It may work better to have an indexed array of branch pointers, or an extendable and variable length list of branch pointers

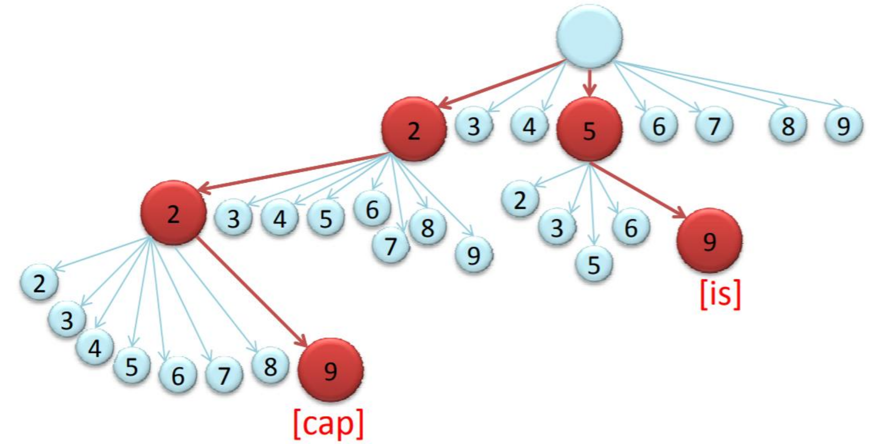
```
struct TrinaryTreeNode {
  char* data;
  struct TrinaryTreeNode* left;
  struct TrinaryTreeNode* middle;
  struct TrinaryTreeNode* right;
}
struct QuadTreeNode {
  char* data;
  struct QuadTreeNode* children[4];
}
```

Prefix Trees (Trie)

- Compact storage
  - Or generative automaton
- Key of each node defined entirely by position
- Compact data storage
- Efficient worst-case searching
- Strings often use 26-ary tree
  - Predictive text
  - Spell check



T9 Trie



Number pad buttons associated with letters – could be used to type characters

- What is the branching factor?
- What does a node contain?
  - What data is in a node?
  - How might we store the branches?

Keypad	Keypad	kepad
1	2 a b c	3 d e f
4 g h i	5 j k l	6 m n o
7 p q r s	8 t u v	9 w x y z
*	0 _	#