Linked Lists

Special Topic

Introduction

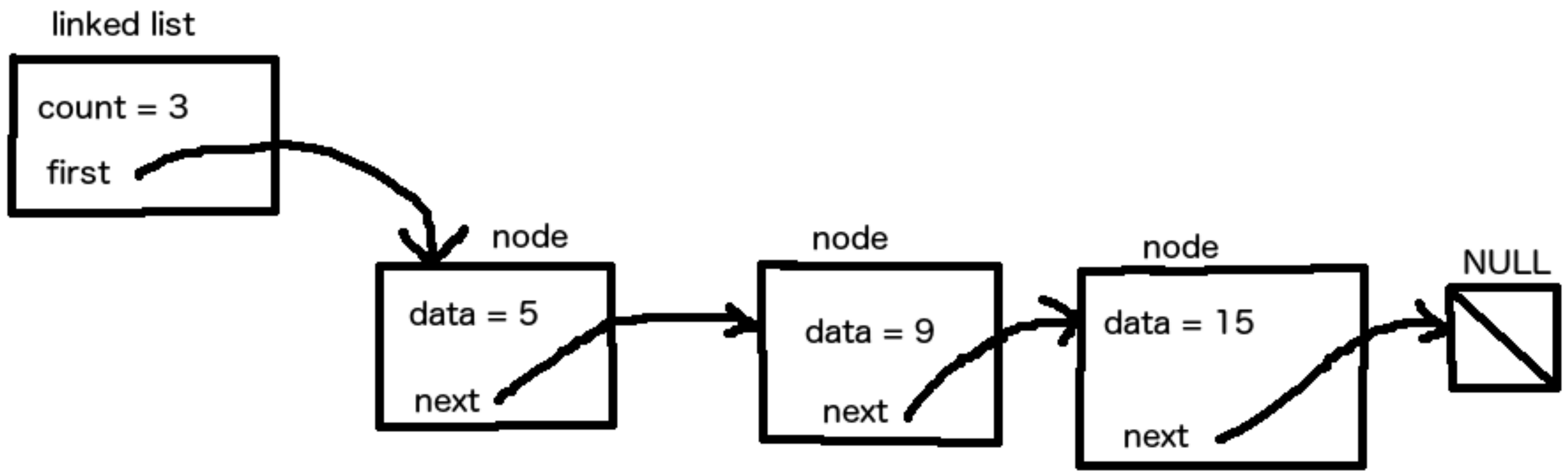
- A major application of pointers and dynamic memory allocations is **data structures**
- One of the most basic data structures is an **array**
- A number of other data structures could be thought of as special variations of arrays

- A downside to an array is that it has a limited size and we have to keep track of its size
- A **list** is a data structure very similar to an array, but it keeps track of its size and automatically resizes as needed
- So, it's like a more convenient array

- A couple variations of lists
  - **Array lists**
  - **Linked lists**
- Each has its own merits and cons
- We'll be looking at **linked lists**
  - Conceptual basis for many other data structures

Linked Lists

- Idea behind a linked list is that we have, instead of elements, **nodes** connected together like a chain
- Typically, a node would simply be a structure variable
- And the link connecting the nodes would be a pointer
- Usually have a linked list structure itself that would contain a pointer to the first node and maybe the count for the number of nodes


linked list
count = 3
first
node
data = 5
next
node
data = 9
next
node
data = 15
next
NULL

```
linked list type
{
    count of nodes in list
    pointer to first node in list
}
```

- Many other data structures are also implemented with this idea of nodes linked together
    - Stacks, queues, trees, etc.

- As the linked list grows, need to use dynamic memory allocation to create new nodes and add to the list
- As the link list shrinks, need to free the dynamically allocated memory
- Also need to be able to access and set nodes inside of the list

- So typical operations a linked list might have are:
  - add() – add a new node to the list
  - remove() – remove a node from the list
  - get() – access the data at a particular node
  - set() – set the data at a particular node
- Could have others, but those most basic
  - init(), print(), etc.

Node

- A node is a structure variable
- Typically a node structure data type will have a member for the data stored in that node and a node pointer to the next node in the list

```
node type
{
    actual data stored in node
    pointer to next node in list
}
```

- Depending on the type of data we plan to store in the list, set the data type of the data member

Example

- **linked_list_test.c**
  - Simple implementation of a linked list for holding integers