DSA Linked Lists in Memory

‹ Previous Next ›

Computer Memory

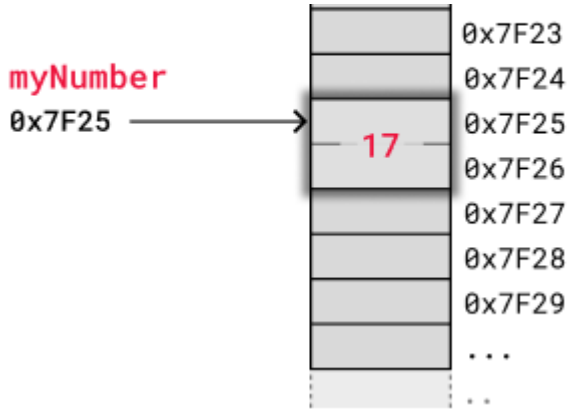
To explain what linked lists are, and how linked lists are different from arrays, we need to understand some basics about how computer memory works.

Computer memory is the storage your program uses when it is running. This is where your variables, arrays and linked lists are stored.

Variables in Memory

Let's imagine that we want to store the integer "17" in a variable `myNumber`. For simplicity, let's assume the integer is stored as two bytes (16 bits), and the address in memory to `myNumber` is `0x7F25`.

`0x7F25` is actually the address to the first of the two bytes of memory where the `myNumber` integer value is stored. When the computer goes to `0x7F25` to read an integer value, it knows that it must read both the first and the second byte, since integers are two bytes on this specific computer.



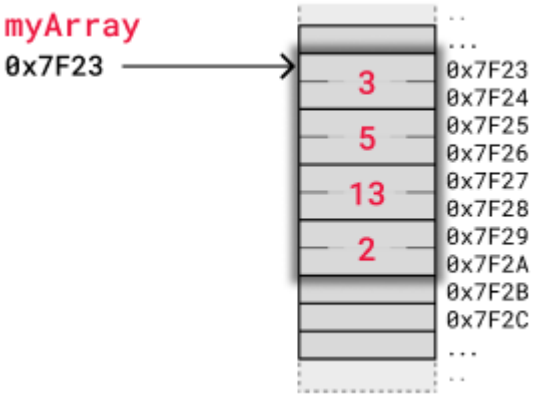
The example above shows how an integer value is stored on the simple, but popular, Arduino Uno microcontroller. This microcontroller has an 8 bit architecture with 16 bit address bus and uses two bytes for integers and two bytes for memory addresses. For comparison, personal computers and smart phones use 32 or 64 bits for integers and addresses, but the memory works basically in the same way.

Arrays in Memory

To understand linked lists, it is useful to first know how arrays are stored in memory.

Elements in an array are stored contiguously in memory. That means that each element is stored right after the previous element.

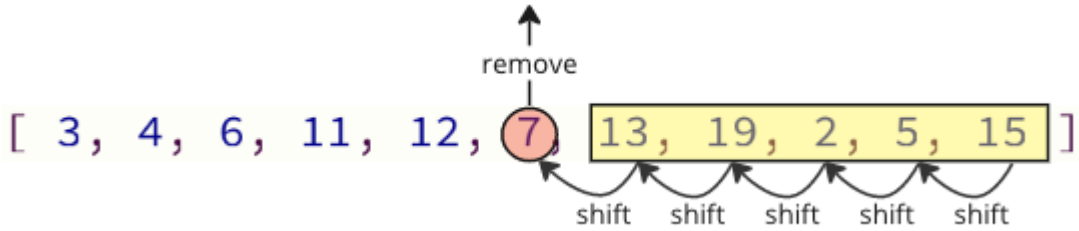
The image below shows how an array of integers `myArray = [3,5,13,2]` is stored in memory. We use a simple kind of memory here with two bytes for each integer, like in the previous example, just to get the idea.



The computer has only got the address of the first byte of `myArray`, so to access the 3rd element with code `myArray[2]` the computer starts at `0x7F23` and jumps over the two

either shifted up to make place for the new element, or shifted down to take the removed element's place. Such shifting operations are time consuming and can cause problems in real-time systems for example.

The image below shows how elements are shifted when an array element is removed.



Manipulating arrays is also something you must think about if you are programming in C, where you have to explicitly move other elements when inserting or removing an element. In C this does not happen in the background.

In C you also need to make sure that you have allocated enough space for the array to start with, so that you can add more elements later.

You can read more about arrays on this previous DSA tutorial page.

Linked Lists in Memory

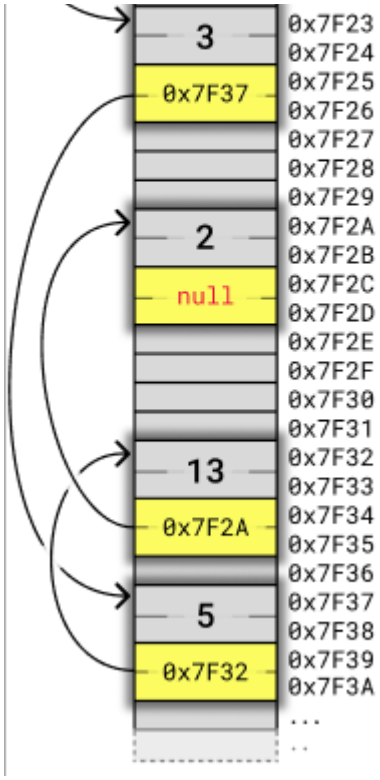
Instead of storing a collection of data as an array, we can create a linked list.

Linked lists are used in many scenarios, like dynamic data storage, stack and queue implementation or graph representation, to mention some of them.

A linked list consists of nodes with some sort of data, and at least one pointer, or link, to other nodes.

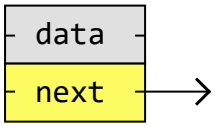
A big benefit with using linked lists is that nodes are stored wherever there is free space in memory, the nodes do not have to be stored contiguously right after each other like elements are stored in arrays. Another nice thing with linked lists is that when adding or removing nodes, the rest of the nodes in the list do not have to be shifted.

The image below shows how a linked list can be stored in memory. The linked list has four nodes with values 3, 5, 13 and 2, and each node has a pointer to the next node in the list.

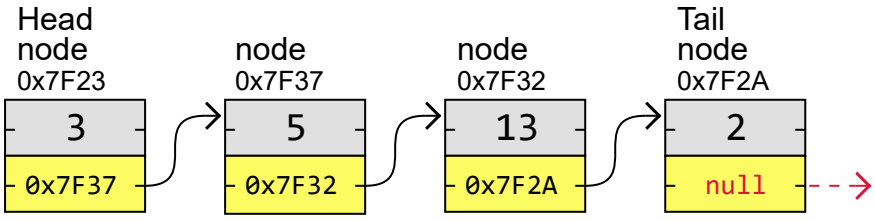


Each node takes up four bytes. Two bytes are used to store an integer value, and two bytes are used to store the address to the next node in the list. As mentioned before, how many bytes that are needed to store integers and addresses depend on the architecture of the computer. This example, like the previous array example, fits with a simple 8-bit microcontroller architecture.

To make it easier to see how the nodes relate to each other, we will display nodes in a linked list in a simpler way, less related to their memory location, like in the image below:



If we put the same four nodes from the previous example together using this new visualization, it looks like this:



This means that when inserting or removing a node, shifting of other nodes is not necessary, so that is a good thing.

Something not so good with linked lists is that we cannot access a node directly like we can with an array by just writing `myArray[5]` for example. To get to node number 5 in a linked list, we must start with the first node called "head", use that node's pointer to get to the next node, and do so while keeping track of the number of nodes we have visited until we reach node number 5.

Learning about linked lists helps us to better understand concepts like memory allocation and pointers.

Linked lists are also important to understand before learning about more complex data structures such as trees and graphs, that can be implemented using linked lists.

Memory in Modern Computers

So far on this page we have used the memory in an 8 bit microcontroller as an example to keep it simple and easier to understand.

Memory in modern computers work in the same way in principle as memory in an 8 bit microcontroller, but more memory is used to store integers, and more memory is used to store memory addresses.

The code below gives us the size of an integer and the size of a memory address on the server we are running these examples on.

Example

Code written in C:

```
#include <stdio.h>

int main() {

    int myVal = 13;

    printf("Value of integer 'myVal': %d\n", myVal);
```

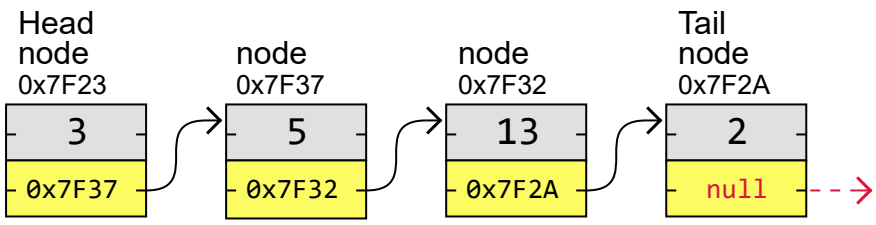
```
    return 0;
}
```

Try it Yourself »

The code example above only runs in C because Java and Python runs on an abstraction level above specific/direct memory allocation.

Linked List Implementation in C

Let's implement this linked list from earlier:



Let's implement this linked list in C to see a concrete example of how linked lists are stored in memory.

In the code below, after including the libraries, we create a node struct which is like a class that represents what a node is: the node contains data and a pointer to the next node.

The `createNode()` function allocates memory for a new node, fills in the data part of the node with an integer given as an argument to the function, and returns the pointer (memory address) to the new node.

The `printList()` function is just for going through the linked list and printing each node's value.

Inside the `main()` function, four nodes are created, linked together, printed, and then the memory is freed. It is good practice to free memory after we are done using it to avoid memory leaks. Memory leak is when memory is not freed after use, gradually taking up more and more memory.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void printList(Node* node) {
    while (node) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("null\n");
}

int main() {
    Node* node1 = createNode(3);
    Node* node2 = createNode(5);
    Node* node3 = createNode(13);
    Node* node4 = createNode(2);

    node1->next = node2;
    node2->next = node3;
    node3->next = node4;

    printList(node1);
```

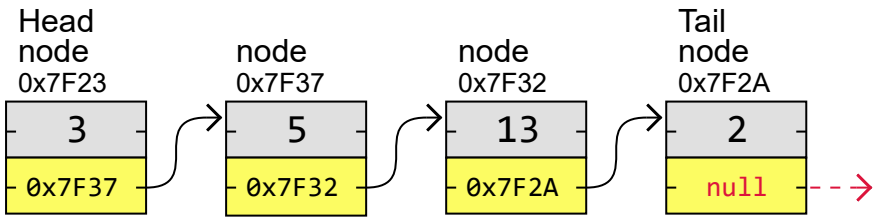
```
    free(node3);
    free(node4);

    return 0;
}
```

Try it Yourself »

To print the linked list in the code above, the `printList()` function goes from one node to the next using the "next" pointers, and that is called "traversing" or "traversal" of the linked list. You will learn more about linked list traversal and other linked list operations on the Linked Lists Oprations page.

Linked List Implementation in Python and Java

We will now implement this same linked list using Python and Java instead.



In the Python code below, the `Node` class represents what a node is: the node contains data and a link to the next node.

The `Node` class is used to to create four nodes, the nodes are then linked together, and printed at the end.

As you can see, the Python code is a lot shorter than the C code, and perhaps better if you just want to understand the concept of linked lists, and not how linked lists are stored in memory.

Example

A basic linked list in Python:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

node1 = Node(3)
node2 = Node(5)
node3 = Node(13)
node4 = Node(2)

node1.next = node2
node2.next = node3
node3.next = node4

currentNode = node1
while currentNode:
    print(currentNode.data, end=" -> ")
    currentNode = currentNode.next
print("null")
```

Try it Yourself »

DSA Exercises

Test Yourself With Exercises

Exercise:

```
A good thing about Linked Lists
is that when inserting or
removing a node, other elements
do not have to be          in memory.
```

Submit Answer »

Start the Exercise

‹ Previous Sign in to track progress Next ›

COLOR PICKER

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
AngularJS Reference
jQuery Reference