Linked Lists

- Linked lists and arrays are similar since they both store collections of data.
- An array allocates memory for all its elements together as one block of memory.
- In contrast, a linked list allocates space for each element separately in its own block of memory called a "linked list element" or "node".
- The best and simplest example of a dynamic data structure that uses pointers for its implementation.
Linked Lists

- A set of dynamically allocated nodes
- The pointers point to the next member of the list

![Diagram of linked lists](image)

Linked Lists

- Iterating over a list
  - no “random” access

![Diagram of linked list iteration](image)

Base pointer
Linked Lists

- Adding a node in the specific position

- Deleting a node in the specific position
Linked Lists

- **Structure of a Node**
  - A value can be any datatypes such as int, float, and struct depending on the data structure of nodes

```c
typedef struct list_el
{
    int value; // value
    struct list_el *next;
}item;
```

Linked Lists

- **Memory Allocation**
Linked Lists

- Memory Allocation

```c
pBase = NULL;
//curr = (item *)malloc(sizeof(item));
curr = new item;
curr->value = nValue;
curr->next = pBase;
pBase = curr;
```

- Iterating over a list

```c
curr = pBase;
while(curr)
{
    printf("%d\n", curr->val);
    curr = curr->next;
}
```
Linked Lists

- Deallocation of memory

```c
curr = pBase;
while(curr)
{
    nextCurr = curr->next;
    free(curr);
    curr = nextCurr;
}
```

Advantages

- Linked lists act like an array, but can grow and shrink as needed.
  - Items can be added or removed from the middle of the list
  - There is no need to define an initial size
  - Memory efficiency
  - Linked lists are non-contiguous; the logical sequence of items in the structure is decoupled from any physical ordering in memory
Disadvantages

- There is no “random” access
- Dynamic memory allocation and pointers are required
- Have a much larger overhead over arrays
  - Items are dynamically allocated
  - Each item must store an additional pointers

Other features

- Dummy Nodes for head or tail
- Doubly Linked
- Circular Linked Lists
Linked list with a dummy head node

- A dummy is used for a head node
  - Has an next pointer, but no value
  - The base pointer is fixed

![Dummy Node Diagram](image)

Doubly Linked

- Each node points to both its predecessor and its successor

![Doubly Linked Diagram](image)
Doubly Linked with Dummy Nodes

- Dummy Nodes for a head and a tail

Circular Linked Lists

- A tail points to a head node to access data circulating the linked list