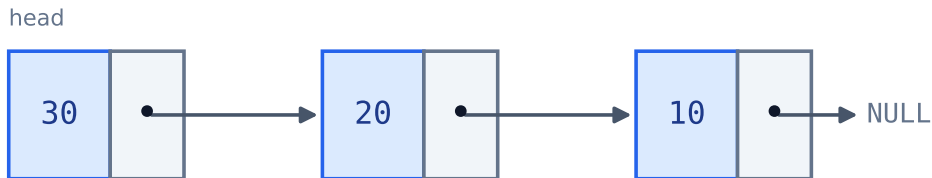


Data structures

C Reference

Linked lists

A linked list is a chain of nodes . Each node holds a value and a pointer to the **next** node; the last one points to **NULL**. Unlike an array it grows one node at a time with **malloc**. Always **free** every node when done.



10-linked-list

A linked list: each node holds a value and a pointer to the next, ending at NULL

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

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

int main(void) {
    Node *head = NULL;
    for (int v = 10; v <= 30; v += 10) { // prepend 10, 20, 30
        Node *n = malloc(sizeof(Node));
        n->value = v;
        n->next = head;
        head = n;
    }
    for (Node *p = head; p != NULL; p = p->next) {
        printf("%d ", p->value); // 30 20 10
    }
    printf("\n");

    while (head != NULL) { // free the whole list
        Node *t = head;
        head = head->next;
        free(t);
    }
    return 0;
}
```

Stacks & queues

A stack is LIFO (last in, first out): push and pop at the same end. A queue is FIFO (first in, first out): add at the back, remove from the front. Both are easy to build on an array with index variables.

```
#include <stdio.h>

int main(void) {
    int stack[10];
    int top = 0;                // next free slot
    stack[top++] = 1;           // push
    stack[top++] = 2;
    stack[top++] = 3;
    while (top > 0) {
        printf("%d ", stack[--top]); // pop: 3 2 1
    }
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    int queue[10];
    int front = 0, back = 0;
    queue[back++] = 1;          // enqueue
    queue[back++] = 2;
    queue[back++] = 3;
    while (front < back) {
        printf("%d ", queue[front++]); // dequeue: 1 2 3
    }
    printf("\n");
    return 0;
}
```

Common mistakes

- In a linked list, never lose the **head** pointer, or the whole list is unreachable.
- Free every node when you are done, walking the list before you unlink.
- Check for an empty list (a NULL head) before you pop.