# CS 201, Fall 2001
Final Discussion Section Notes

## 1 Keeping Linked List Information

When working with lists, there seem to be certain items of information about the list that is common. Some of these things are:

- Number of nodes in a list
- A total of some values in the list
- The head of the list
- Sometimes we need to know the tail of the list
- etc.

It is quite common to have another type of struct that holds this information about the list. It can increase efficiency in the program by always keeping a total of some data members, etc.

Here's an example:

```c
typedef struct node *NODEPTR;

typedef struct node {
    int data;
    NODEPTR next;
} NODE;

/* Struct to keep info about the list */
typedef struct listInfo {
    int numNodes;
    int dataTotal;
    NODEPTR head;
    NODEPTR tail;
} LISTINFO;
```

## 2 Queue

A queue is an ADT in which items are inserted into the front of the list and removed from the back. In real life it is basically a line, such as those you get into when you want to buy something in a store. A queue is commonly referred to as a FIFO data structure: First In, First Out.
2.1 Fundamental Operations

- Enqueue - add an item to the queue
- Dequeue - remove an item from the queue

2.2 Simple Array Implementation

It is quite possible to use an array to implement a queue, but we would suffer all the same space limitations that we have all along with arrays.

We would have to guess about how big the queue could possibly get, declare an array to be of that size, and then start enqueueing items. We would just have to hope that our guess was correct, so that the array could hold the entire queue. Otherwise we’d have to abort the program.

To implement the queue simply maintain a count of the number of elements into the array.

Enqueueing is not a big problem, we would just put a value into the first "empty" element of the array however dequeuing is somewhat of a problem. Since when we dequeue something, it has to be removed from the beginning of the queue.

2.3 Linked list Implementation

Since we now know about linked lists, we should consider using a linked list implementation of the queue. The queue can be implemented without wasting any space, and by keeping pointers to both the beginning of the list (the head) and the end of the list (the tail), we can make our queue work much faster than an array implementation.

3 Stack

A stack is an ADT in which items are inserted and removed from the front of a list. In real life a good example of a stack would be a stack of trays in a cafeteria. In a stack the first item inserted is the last item removed. Thus a stack is commonly referred to as a LIFO data structure: Last In, First Out.

3.1 Fundamental Operations

- Push - add an item to the stack
- Pop - remove an item from the stack

3.2 Implementation

Array or Linked list.

Implement it using a linked list by always inserting and removing from the front of the list.