


# Module 17 Linked Lists

## 1. Module 17

### 1.1 CS Bridge: Linked Lists




NYU TANDON  
ONLINE

CS Bridge: Linked Lists


Module 17  
Dan Katz

### 1.2 In this module



In This Module

- What is a linked list
- Why do we need for linked lists
- Working with templates
- How is a linked list designed
- What are linked lists used for

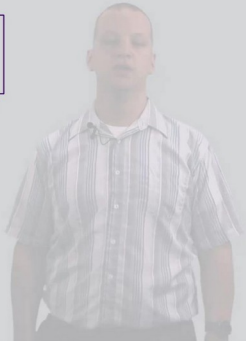
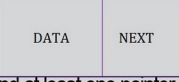


## 1.3 What is a linked list

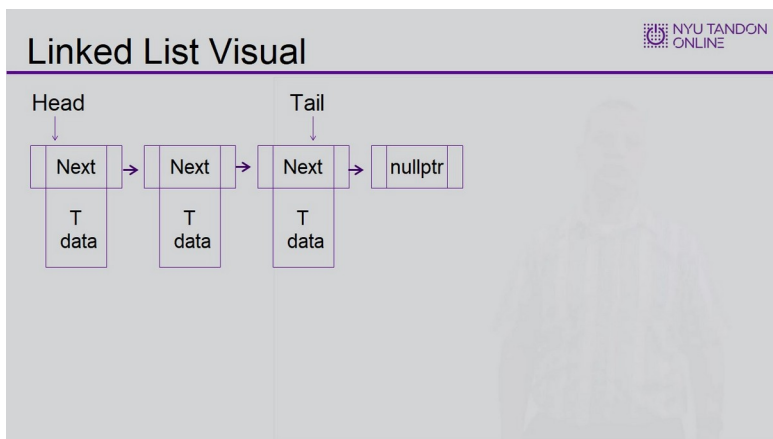
### What is a Linked List

NYU TANDON ONLINE

- Linked lists are a fundamental data structure
- In a list data is stored in a node
- Each node contains a data section and at least one pointer (next)
- Data stores just one item of the list
- The pointers (next) are connected to form a chain of nodes
- The list is recorded by the head, a pointer to the first node
- The last node pointer will point to nullptr.



## 1.4 Linked list visual



### Notes:

<http://www.cs.usfca.edu/~srollins/courses/cs112-f08/web/notes/linkedlists/ll2.gif>

## 1.5 Why do we need linked lists

### Why Do We Need Linked Lists?



- Whereas arrays can access any element in  $O(1)$ , Linked list access is  $O(N)$
- However, array insertion is  $O(N)$ , where linked list is  $O(1)$
- Linked Lists can be reorganized, merged, etc, easily
- Linked Lists do not require any overhead for storage, aside from the pointers

## 1.6 Working with templates

### Working With Templates



- In C++ a class can be templated when we don't know the data type of one or more elements.
- Before each function or class, we put "template <class T>"
- C++ replaces "T" in that function with an actual data type when it knows what data type it would be.

```
template <class T> //template <typename SomeName>
void mySwap(T& a, T& b){
    T temp = a;
    a = b;
    b = temp;
}
```

## 1.7 Templated classes

### Templated Classes



- When a class is templated, its name changes to include the data type which is templated
- In main, this class would be created as  
    SomeVal<int> varName;  
    SomeVal<char> varName;

```
template <class T>
class SomeVal{
    T data;
public:
    T getVal() const{ return data; }
    void setVal(T newValue);
};

template <class T>
void SomeVal<T>::setVal(T newValue){
    data = newValue;
}
```

Note the change in name

## 1.8 Knowledge Check

(Multiple Response, 10 points, 1 attempt permitted)

Knowledge Check

NYU TANDON  
ONLINE

For the following code sample, what are acceptable data types for the parameters, a and b? (Select all that apply)

```
template <class T>
void mySwap(T&a, T&b){
    T temp = a;
    a = b;
    b = temp;
}
```

☒ int, int  
☐ int, string  
☐ char, int  
☒ double, double

Correct	Choice
X	int, int
	int, string
	char, int
X	double, double

### Feedback when correct:

That's right! You selected the correct response.

### Feedback when incorrect:

Remember that you cannot have two different data types when calling in a templated function!

## Correct (Slide Layer)

### Knowledge Check

For the following code sample, what are acceptable data types for the parameters, a and b? (Select all that apply)

```
template <class T>
void mySwap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```

Correct

That's right! You selected the correct response.

Continue

## Incorrect (Slide Layer)

### Knowledge Check

For the following code sample, what are acceptable data types for the parameters, a and b? (Select all that apply)

```
template <class T>
void mySwap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```

Incorrect

Remember that you cannot have two different data types when calling in a templated function!

Continue

## 1.9 Designing a linked list Node

### Designing a Linked List Node

- Nodes should be templated since we don't know what datatype will be stored in them.

```
template <class T>
class LListNode{
    T data;
    LListNode<T>* next;
public:
    LListNode(T newdata = T(), LListNode<T>* newNext = nullptr) :
        data(newdata), next(newNext) {}
    friend class LList< T > ;
};
```

Item to be stored

Pointer to the next node

Friend to make life easier

Constructor to make life easier

## 1.10 Designing a linked list

### Designing a Linked List

NYU TANDON ONLINE

```
template <class T>
class LList{
    LListNode<T>* head;
    LListNode<T>* recursiveCopy(LListNode<T>* rhs);
public:
    LList() :head(nullptr){}
    LList(const LList& rhs) :head(nullptr) { *this = rhs; }
    LList<T>& operator=(const LList<T>& rhs);
    ~LList(){ clear(); }
    void insertAtHead(T newdata);
    T removeFromHead();
    bool isEmpty()const { return head == nullptr; }
    void clear();
    void insertAtEnd(T newdata);

    void insertAtPoint(LListNode<T>* ptr, T newdata);
    int size() const;
};
```

The only item stored in the list class is a pointer to the first node in the list!

## 1.11 Stopping at the end vs. going off the end

### Stopping at the end vs. going off the end

NYU TANDON ONLINE

- A common problem in understanding lists is the misunderstanding of stopping at the end versus going off the end.
- Both solutions use a while, but with different conditions
- Stopping at the end: while(temp->next != nullptr)
- Going off the end: while(temp!=nullptr)

**Solution 1**

**Solution 2**

### Solution 2 (Slide Layer)

### Stopping at the end vs. going off the end

NYU TANDON ONLINE

- A common problem in understanding lists is the misunderstanding of stopping at the end versus going off the end.
- Both solutions use a while, but with different conditions
- Stopping at the end: while(temp->next != nullptr)
- Going off the end: while(temp!=nullptr)

**Solution 1**

**Solution 2**

```
template <class T>
void LList<T>::insertAtEnd(T newdata) {
    if (isEmpty()) {
        insertAtHead(newdata);
        return;
    }
    LListNode<T>* temp = new
    LListNode<T>(newdata);
    LListNode<T>* end = head;
    while (end->next!= nullptr)
        end = end->next;
    end->next = temp;
}
```

## Solution 1 (Slide Layer)

### Stopping at the end vs. going off the end

- A common problem in understanding lists is the misunderstanding of stopping at the end versus going off the end.
- Both solutions use a while, but with different conditions
- Stopping at the end: while(temp->next != nullptr)
- Going off the end: while(temp!=nullptr)

**Solution 1**

**Solution 2**

```
template <class T>
int LList<T>::size() const{
    int count = 0;
    LListNode<T>* temp = head;
    while (temp != nullptr){
        count++;
        temp = temp->next;
    }
    return count;
}
```

## 1.12 Knowledge Check

(Drag and Drop, 10 points, 4 attempts permitted)

### Knowledge Check

When approaching the end of a linked list, identify the two methods that can be used based on the code sample below :

Stopping at the end

Going off the end

```
while (end-> next != nullptr)
end = end->next;
end->next= temp;
```

```
while (temp != nullptr)
    count++;
    temp = temp->next;
```

Drag Item	Drop Target
Stopping at the end	while (end-> next != nullptr)  end = end->next;  end->next= temp;
Going off the end	while (temp != nullptr)  count++;

```
temp = temp->next;
```

Drag and drop properties

Return item to start point if dropped outside the correct drop target

Snap dropped items to drop target (Stack random)

Delay item drop states until interaction is submitted

**Feedback when correct:**

That's right! You selected the correct response.

**Feedback when incorrect:**

You did not select the correct response.

**Correct (Slide Layer)**

Knowledge Check

NYU TANDON  
ONLINE

When approaching the end of a linked list, identify the two methods that can be used based on the code sample below :

Stop

Correct

That's right! You selected the correct response.

Continue

```
while (end->next != nullptr)
end = end->next;
end->next = nullptr;
```

```
while (temp != nullptr)
count++;
temp = temp->next;
```



## Incorrect (Slide Layer)

NYU TANDON  
ONLINE

### Knowledge Check

When approaching the end of a linked list, identify the two methods that can be used based on the code sample below :

Stop

Incorrect  
You did not select the correct response.

Continue

```
while (end->next != nullptr)
    end = end->next;

while (temp != nullptr)
    count++;
    temp = temp->next;
```

## Try Again (Slide Layer)

NYU TANDON  
ONLINE

### Knowledge Check

When approaching the end of a linked list, identify the two methods that can be used based on the code sample below :

Stop

Incorrect  
That is incorrect. Please try again.

Try Again

```
while (end->next != nullptr)
    end = end->next;

while (temp != nullptr)
    count++;
    temp = temp->next;
```

## 1.13 Recursion in lists

NYU TANDON  
ONLINE

### Recursion in Lists

- Recursion is often used in linked lists because a sub list looks the same as the larger list
- Here is a recursive copy operation done with the constructor for the LListNode class we created

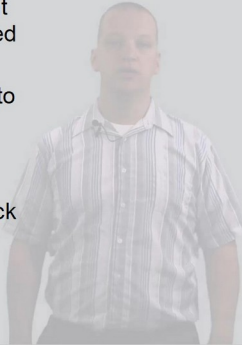
```
template <class T>
LListNode<T>* LList<T>::recursiveCopy(LListNode<T>* rhs) {
    if (rhs == nullptr)
        return nullptr;
    return new LListNode<T>(rhs->data, recursiveCopy(rhs->next));
}
```

## 1.14 What are linked lists used for

### What Are Linked Lists Used For



- Anywhere we need storage with constant time insertion, no overhead but don't need anything other than linear access
- In a later module you will use linked lists to develop stacks and queues, other data structures
- Used in the FAT32 file system to save data on a hard drive (each hard drive block contains a pointer to the next block; the FAT contains only a pointer to the first block)



## 1.15 In this module

### In This Module



- What is a linked list
- Why do we need for linked lists
- Working with templates
- How is a linked list designed
- What are linked lists used for



## 1.16 End of Module



End of Module

Exit

