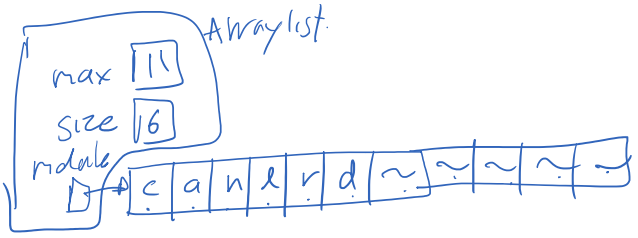
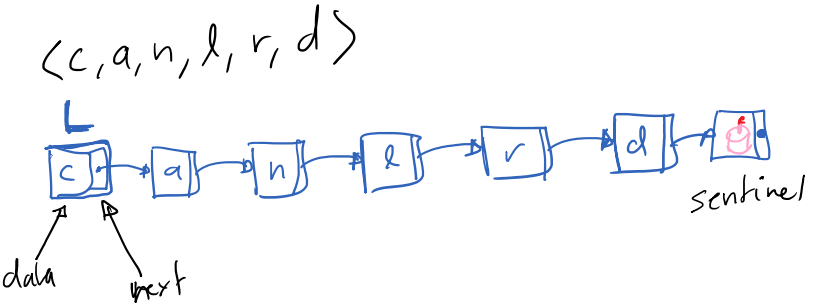
The 'LinkedList' Data Structure

Friday, September 20, 2019 5:11 PM

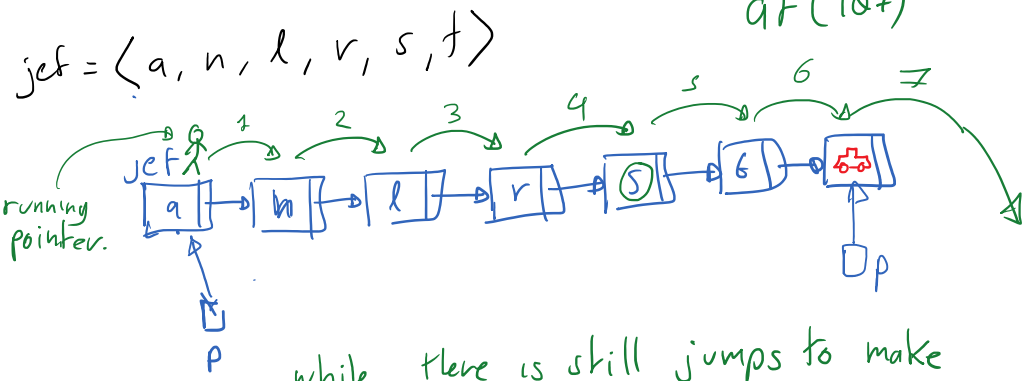





```
template <typename T>
class LinkedList {
   T m_data;
   LinkedList* m_next;
… …
};
```

LinkedList Bob:

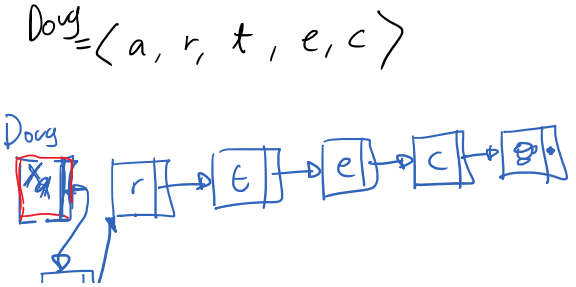
at(i)




```
T& at( int i )
{
   LinkedList* p = this;
   int k = 0;

   while( k<i && p->m_next != NULL ){
      p = p -> m_next;
      k++;
   }

   if( p->m_next == NULL )
      error();

   return p->m_data;
}
```

while there is still jumps to make and we have not reach the sentinel.

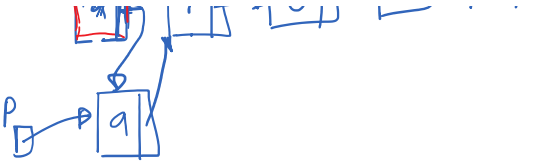
Doug.insert_front(x)



1.- Create new node

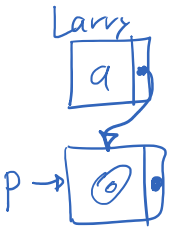
2.- Copy data to new container

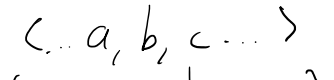
1.- Create new node
2. Copy data to new container
3.- Insert new element
4.- Set new node pointer
5. Set calling object pointer



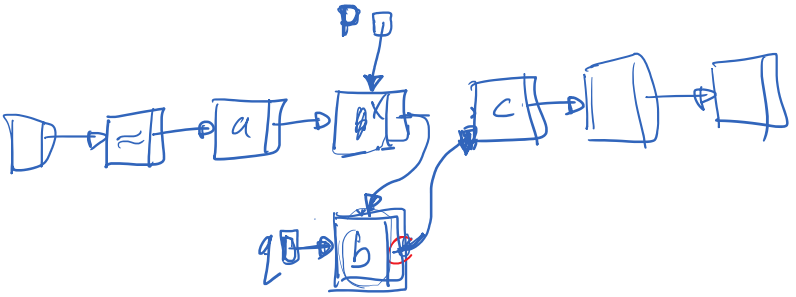
Larry . insert_front (a)

```
LinkedList::insert_front(T& x)
{
   LinkedList *p = new LinkedList;
   p->m_data = m_data;
   m_data = x;
   p->m_next = m_next;
   m_next = p;
}
```



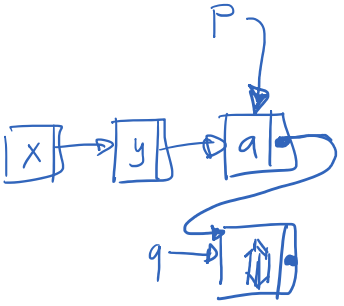


insert(LinkedList* p, X)



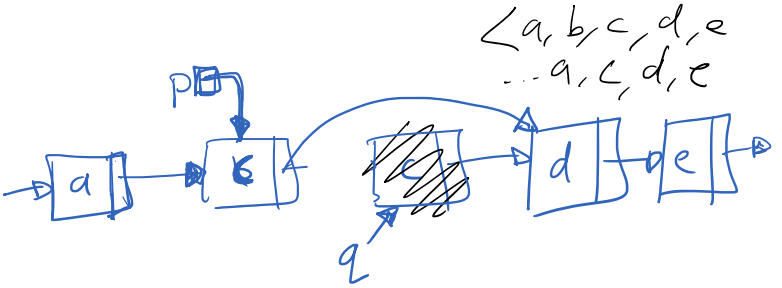
1 make a new node
2 copy data
3 insert new element
4 set pointer of new container
5 set pointer at position.

```
LinkedList::insert(LinkedList* p, T& x)
{
   LinkedList* q = new LinkedList;
   q->m_data = p->m_data;
   p->m_data = x;
   q->m_next = p->m_next;
   p->m_next = q;
}
```



remove at p

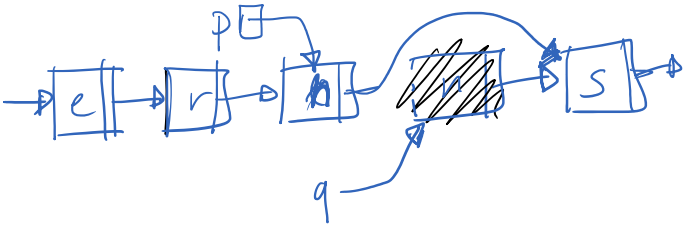
1 copy next element into p's element
2 make second pointer q point to p's next
3 change p's next to bypass node



2 [illegible]

3 change p's next to bypass node

4 delete q




```
LinkedList::remove(LinkedList* p)
{
   p->m_data = p->m_next->m_data;
   LinkedList *q = p->m_next;
   p->m_next = q->m_next;
   delete q;
}
```