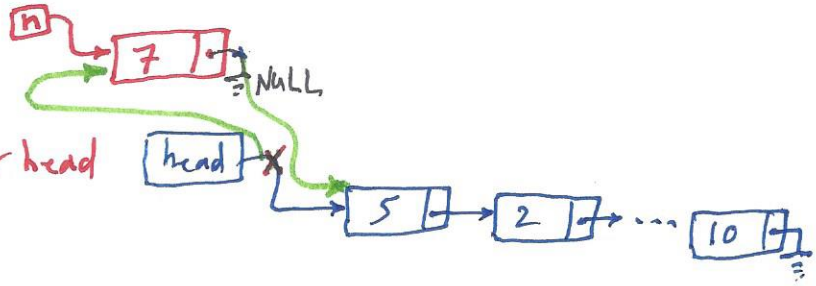
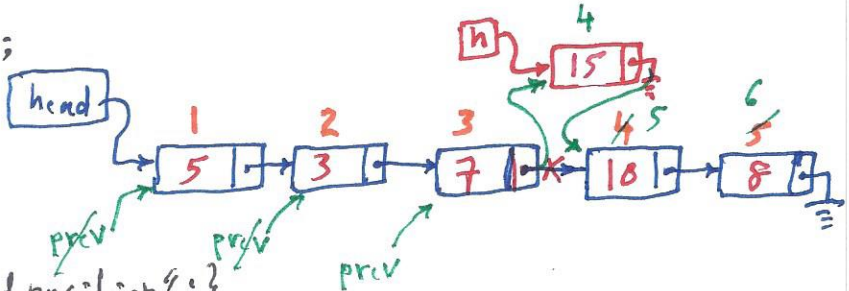
Week 7, Sat:

```
// To insert a Node at a given position
void List:: insertAtPos (int value, int pos) // pos should be a positive integer
{
   Node *n = new Node;
   n->data = value; // 7
   n->next = NULL;
   if (pos == 1) // insert after head
   { n->next = head;
     head = n;
   } else   // pos = 4
        { Node *prev = head;
          int Count = 1;
          while (Count < pos-1 && prev != null)
          {   prev = prev->next;
              Count++;
          }
          if (prev == NULL)
            { Cout << "Invalid position";}
            else { n->next = prev->next;
                   prev->next = n;
                 }
          }//endelse
   }// end insertAtPos
   // list1.insertAtPos (15, 4);
```

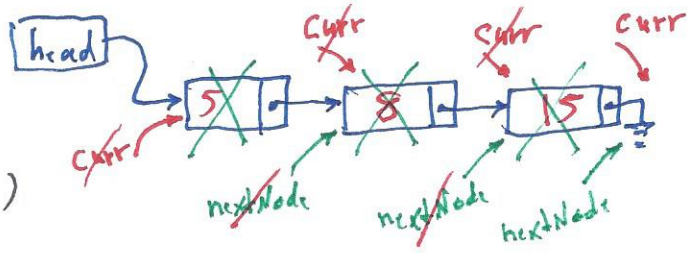




(data, pos)

- Destroying a List: Destructor

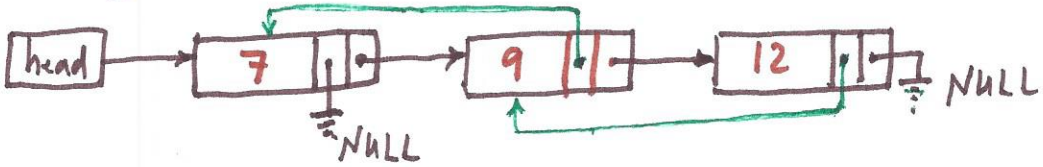
```
List:: ~List ( ) //destructor, we need to traverse the list and delete each node.
{  Node *curr;
   Node *nextNode;
   Curr = head;
   while (Curr != NULL)
   {
```



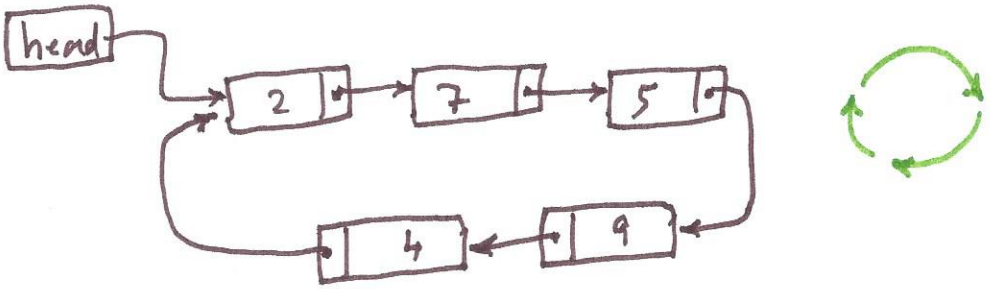
```
        nextNode = Curr->next;
        delete Curr;
        Curr=nextNode;
    }
    head=NULL; //
} //end destructor
```

- Different Linked List:

1- Doubly Linked List



2- Circular Linked List:



Function Templates (Generic Function):

- A generic function defines a set of general operations that will be applied to various type of data (data types).
- A generic functron has the type of data that it will operate upon passed to it as a parameter. For example, the Quick sort Alg. is the same whether it is applied to array of integers or an array of float values.

When we are creating a generic func., we are creating a func. that can automatically overload itself.

```
template < class  Ttype >          // or typename
return_type   fun_name (parameter list)
{
        // body of func.
}
```

Ex#1,

```
#include<iostream>
using . . . ;
template <class T>
void swap (T &a , T &b)
{  T temp = a;
     a = b;
     b = temp;
} // end swap
int main()
{  int i=10, j=20;
   double  x=2.7, y=8.3;
   swap(i,j); // swap integers
   swap (x,y); //   "   doubles
   Cout << "Swapped i,j: " << i <<j<<endl;
   Cout << "Swapped x,y: " << x <<y<<endl;
   return 0;
} //end main
```

```
template <class X>          → or typename
void swap ( X &a, X &b)     → No Code
{   . . .
    : : :
}
```

Ex#2, Find the Max of two values:

```
template <typename T>
T findMax (T x, T y)
{
   return (x>y) ? x : y;
}
```

We can declare more than one generic data type with the template statement.

```
#include <iostream>
using - - -  ;
```

```
template < class T1 , class T2 >
void myfunc ( T1 a , T2 b)
{  cout << a << " " << b << endl;
}
int main()
{           int   string
   myfunc ( 10, "Hi");
           double  long int
   myfunc (0.74, 10L );
   return 0;
} // end main
```

- A generic function in C++ is a function template that can operate on different data types without being rewritten for each type.
  It allows for type-independent programming by letting the compiler generate the necessary code to handle the specific type(s) used when the function is called.
  This is achieved using template parameters, which acts as placeholders for any data type.

Overloading Functions:

When funcs. are overloaded, we can have different actions performed with the body of each function. But generic funcs. must perform same general action for all versions.

We can explicitly overload a generic function.

If we overload a generic func., that overloaded func. overrides (or "hides") the generic func relative to that specific version.

Example:

```
#include <iostream>
using . . . ;
template < class X >
void Swap ( X &a, X &b)
{  X temp;
   . . .
}
```

```
// This overrides the generic version of swap
void swap (int a , int b)
{  cout << " This is inside swap(int,int)\n ";
}
int main()
{  int i=10, j=20;
   float x=2.4, y=3.7;
   swap( i , j ); // calls overload swap
   swap(x, y); // calls generic func. swap , floats
   return 0;
} //end main
```

Class Activity:

```
int Square (int num)
{  return num * num;
}

double Square (double num)
{  return num *num;
}
```

Write a generic function for above functions.

- Class Templates (Generic class):

```
template < class T >
class  class_name
{
   . . .
   . . .
}
```

A Linked List template: We can convert the class to a template that can accept any data type.

```
template < class T >
class LinkedList
{  private:  struct Node
            {  T data;
               Node *next;
            }
```

```
        Node *head;
public:
        LinkedList() // default constructor
        { head = NULL; }
       ~LinkedList(); // prototype
        void appendNode(T); // "
        void insertNode(T); // "
        void insertAtPos(T, X); // "
        void deleteNode(T);   // "
        void display() const; // "
}; //end class
template <class T>
void LinkedList<T>::appendNode(T value)
{   . . .
    . . .
}
template <class T>
void LinkedList<T>::display()
{   . . .
    . . .
}
 . . .
  . . .
```