# Lecture 17.1

Topics
1. Linked List Operations – Insertion, Removal, Display, etc.
2. Implementation – List of `int`'s
3. `Fraction` Data – Introduction

___________________________________________________________

## 1. Linked List Operations – Insertion, Removal, Display, etc.

Specific data and selection of notations are discussed for implementation.

Discussions are given in class.

## 2. Implementation – List of `int`'s

Actual sample implementation and sample code are performed in class – see written code for `insertion` and others.

Implementation for removal and other utilities are commented and written in class "**on the go**".

A listing of samples of code/type/typing is given below.

```
struct IntNode {
    int data;
    struct IntNode* next;
};

//struct IntNodeA {
//    int data;
//    struct IntNodeA next; // BAADD
//};

// Typing Examples

typedef struct IntNode TIntNode;
typedef struct IntNode SIntNode;
typedef struct IntNode TSIntNode;

typedef struct IntNode* TIntNodePtr;
typedef struct IntNode* TIntNodeAddr;

typedef struct IntNode TINode;
typedef struct IntNode SINode;
typedef struct IntNode TSINode;

typedef struct IntNode* TINodePtr;
typedef struct IntNode* TINodeAddr;

// The following prefix of "Td" will be used

typedef struct IntNode TdIntNode;
typedef struct IntNode* TdIntNodePtr;
typedef struct IntNode* TdIntNodeAddr;
```

And, a sample setup and the use of the some of the types plus functions are given below.

```
// Application Driver
int main() {
  struct IntNode* head = NULL; //TIntNode* head = NULL;
  TdIntNodePtr tmpPtr = NULL;
  int tmp;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));

  //printf("\nEnter an int:");
  //scanf_s("%d", &tmp);
  //tmpPtr->data = tmp;

  tmpPtr->data = 5;
  tmpPtr->next = head;

  head = tmpPtr;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));
  tmpPtr->data = -50;
  tmpPtr->next = head;

  head = tmpPtr;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));
  tmpPtr->data = -500;
  tmpPtr->next = head;

  head = tmpPtr;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));
  tmpPtr->data = 600;
  tmpPtr->next = head;

  head = tmpPtr;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));
  tmpPtr->data = 700;
  tmpPtr->next = head;

  head = tmpPtr;

  tmpPtr = (TdIntNode*)malloc(sizeof(TdIntNode));
  tmpPtr->data = 800;
  tmpPtr->next = head;

  head = tmpPtr;

  printBackward(head);

  empty(&head);

  return 0;
}
```

```
// Function Definitions

void printBackward(TdIntNodeAddr myList) {

    // TODO
}


void empty(TdIntNodeAddr* myListAddr) {

    // TODO
}
```

## 3. Fraction Data – Introduction

**Definition:**

- **`Fraction` data/value/object must be declared as a `struct Fraction` of two elements, which are integers of `num` and `denom`. Of course, `denom` cannot be ZERO.**
- **These `Fraction` objects must have their negativity to be taken to the numerators (i.e., `num`).**

Again, **Figure 1** shows a sample series of steps in inserting a node to a list.

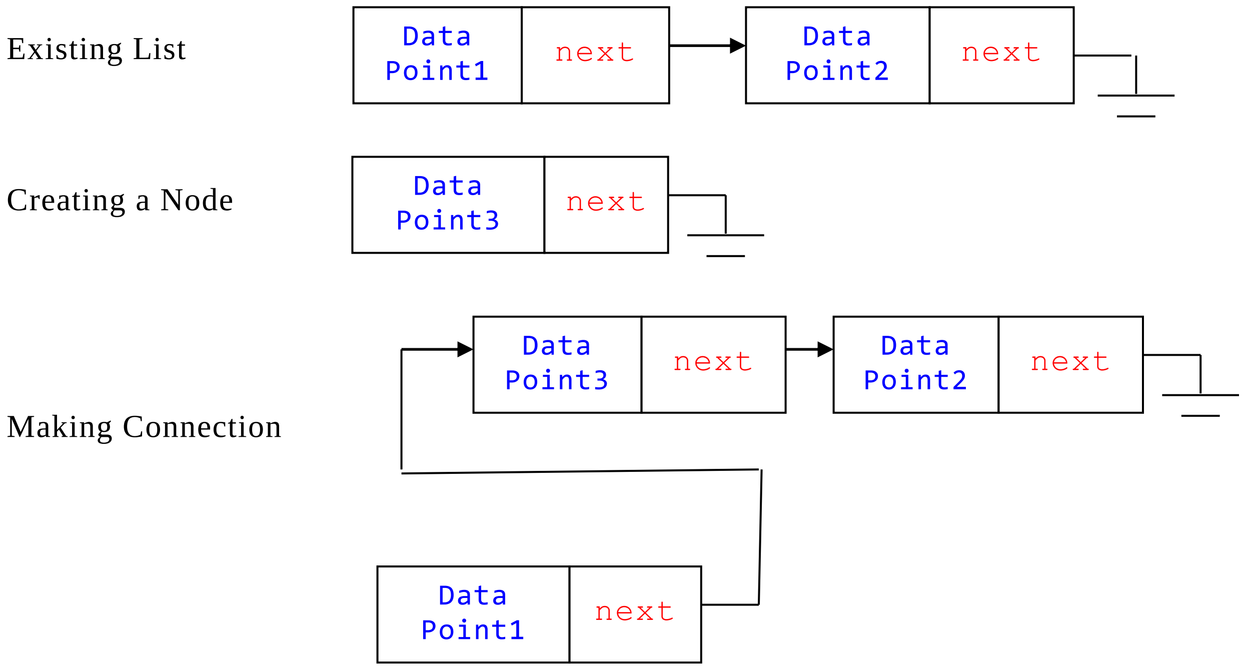

**Figure 1** Steps in inserting a node to an existing list

And **Figure 2** below shows a function `removeFirst()` that will remove the first node in a given list.

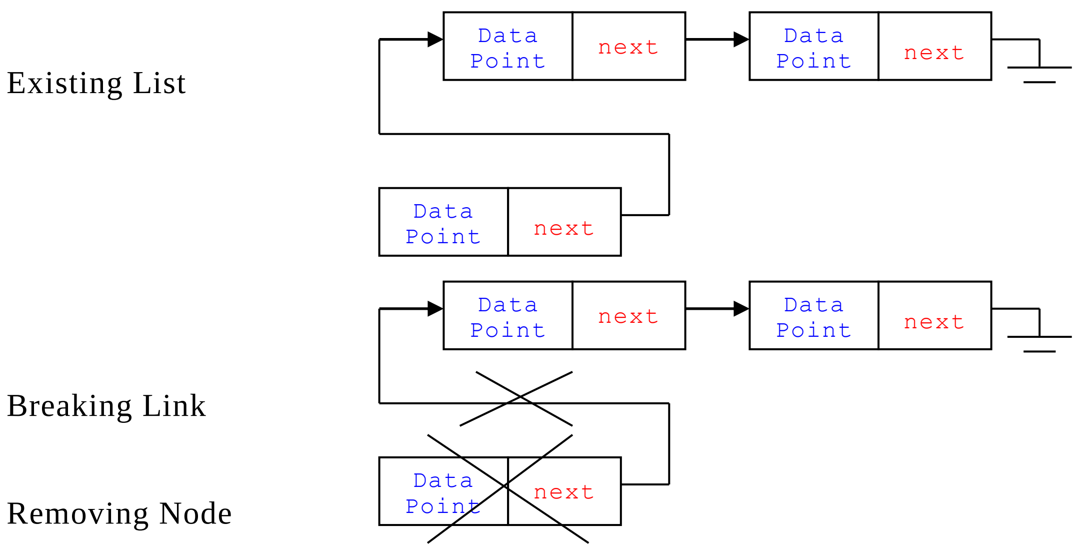

**Figure 2** Steps in removing the first node

Discussion for **`struct Fraction`** implementation will be given in class.

```
struct Fraction {
  int num;
  int denom;
};

struct FractionNode {
  struct Fraction data;
  struct FractionNode* next
};
```

For class discusions, a listing of samples of code/type/typing is given here.

```
struct Fraction {
  int num;
  int denom;
};

typedef struct Fraction TdFraction;

struct PolyTerm {
  int expo;
  TdFraction coeff;
};

typedef struct PolyTerm TdPolyTerm;

struct PolyNode {
  TdPolyTerm data;
  struct PolyNode* next;
};

typedef struct PolyNode TdPolyNode;
```