SINGLY LINKED LISTS

: INSERTION

```
Algorithm traversing() {
 1.new1 = Start
 2.while(new1 != NULL)
           2.1 Print  new1 -> info
           2.2 new1 = new1 -> next
 }
```

```
Start=NULL
Algorithm InsertAtBEG() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info = data]
3. If (Start == NULL)
            3.1 new1 -> next = NULL
            3.2 Last=new1
            3.3 Start=new1
   else
                3.1 new1 -> next=Start
                3.2 Start = new1
}
```

```
Start=NULL
Algorithm InsertAtEnd() {
1. Create node [(new1 = (struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info =d ata]
3.if(Start == NULL)
          3.1 Last=new1
           3.2 Start=new1
else
          3.1 Last -> next = new1
          3.2 Last = new1
4. Last ->next = NULL
}
```

```
Algorithm InsertAtSpec() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)
          4.1 new1 -> next = Start
          4.2 Start = new1
    Else
          4.1 Previous = Start
          4.2 Count = 1
 4.3 While( Count <= Location - 1 && Previous->next !=NULL)
          4.3.1 Previous = Previous -> next
          4.3.2 Count++
4.4 new1 -> next = Previous -> next
4.5 Previous -> next = new1
}
```

```
Algorithm InsertAtSorted() {
1. Enter Data
2. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
3. new1->info=Data
4. If (Data <= Start -> info)
        4.1 new1 -> next = Start
        4.2 Start = new1
Else
        4.1 Current = Start
        4.2 Previous = NULL
4.3While( Data >= Current -> info && Current  != NULL)
        4.3.1 Previous = Current
        4.3.2 Current = Current -> next
4.4 new1 -> next = Current
4.5 Previous -> next = new1
 }
```

: DELETION

```
Algorithm DeleteAtBeg() {
 1. If (Start == NULL)
        1.1 Print "underflow“
else
        1.1 Current = Start
        1.2 Start = Start -> next
        1.3 Current -> next = NULL
        1.4 Release the memory [ free (Current) ]
}
```

```
Algorithm DeleteAtEnd() {
   1. If (Start == NULL)
        1.1 Print "underflow“
    else
            If (Start -> next == NULL )
                1.1 Release the memory [ free (Start) ]
                1.2 Start == NULL
             else
                1.1 Current = Start
                1.2 while ( Current -> next != Last )
                        1.2.1 Current = Current -> next
                1.3 Current -> next = NULL
                1.4 Release the memory [ free (Last) ]
                1.5 Last = Current
}
```

```
Algorithm DeleteAtSpec() {
1. Enter the Location
2. Current = Start
3. Previous = NULL
4. If (Start == 0)
        4.1 Print "underflow“
else
If ( Location == 1)
        4.1 Start = Start -> next
        4.2 Current -> next = NULL
        4.3 Release the memory [ free (Current) ]
else
        4.1 for (i=1 to Location-1)
                4.1.1 Previous = Current
                4.1.2 Current = Current -> next
        4.2 Previous -> next = Current -> next
        4.3 Current -> next = NULL
        4.4 Release the memory [ free (Current) ]
}
```

CIRCULAR LINKED LISTS

```
Algorithm traversing() {
  1.new1 = Last -> next
  2.while(new1 != Last)
          2.1 Print  new1 -> info
          2.2 new1 = new1 -> next
  3. Print Last -> info
 }
```

: INSERTION

```
Last=NULL
Algorithm InsertAtBEG() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info = data]
3. If (Last == NULL)
          3.1 new1 -> next = new1
          3.2 Last=new1
else
          3.1 new1 -> next = Last -> next
          3.2 Last -> next = new1
 }
```

```
Algorithm InsertAtSpec() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)
          4.1 new1 -> next = Last -> next
          4.2 Last -> next = new1
   Else
          4.1 Previous = Last -> next
           4.2 Count = 1
          4.3 While( Count <= Location - 1)
                    4.3.1 Previous = Previous -> next
                     4.3.2 Count++
          4.4    if (Prev == Last)
                              4.4.1 new1 -> next = Last -> next
                              4.4.2 Last ->next=new1;
                              4.4.3 Last = new1;
                    else
                              4.4.1 new1 -> next = Previous -> next
                              4.4.2 Previous -> next = new1
}
```

```
Last=NULL
Algorithm InsertAtEnd() {
1. Create node [(new1 = (struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info =data]
3.if(Last == NULL)
        3.1 new1 -> next = new1
        3.2  Last=new1
else
        3.1 new1 -> next = Last -> next
        3.2 Last -> next = new1
        3.3 Last = new1
}
```

: DELETION

```
Algorithm DeleteAtBeg() {
1. If (Last == NULL)
         1.1 Print "underflow“
else If (Last -> next == Last )
        1.1 Release the memory [ free (Last) ]
        1.2 Last == NULL
else
        1.1 Current = Last -> next
        1.2 Last -> next = Current -> next
        1.3 Current -> next = NULL
        1.4 Release the memory [ free (Current) ]
}
```

```
Algorithm DeleteAtSpec() {
1. Enter the Location
2. Current = Last -> next
3. Previous = NULL
4. If (Last == NULL)
        4.1 Print "underflow“
  else If ( Location == 1)
        4.1 Last -> next = Current -> next
        4.2 Current -> next = NULL
        4.3 Release the memory [ free (Current) ]
  else
        4.1 for (i=1; i<Location; i++)
                4.1.1 Previous = Current
                4.1.2 Current = Current -> next
        4.2 if ( Current == Last)
                4.2.1 prev -> next = Current -> next
                4.2.2 Last = Prev
        else
                4.2.1 Previous -> next = Current -> next
        4.3 Current -> next = NULL
        4.4 Release the memory [ free (Current) ]
 }
```

```
Algorithm DeleteAtEnd() {
1. If (Last == NULL)
        1.1 Print "underflow“
else If (Last -> next == Last )
        1.1 Release the memory [ free (Last) ]
        1.2 Last == NULL
else
        1.1 Current = Last -> next
        1.2 while ( Current -> next != Last )
                1.2.1 Current = Current -> next
        1.3 Current -> next = Last -> next
        1.4 Last -> next = NULL
        1.5 Release the memory [ free (Last) ]
        1.6 Last = Current
}
```

DOUBLY LINKED LISTS

TRAVERSING

```
Algorithm fwdtraversing() {
        1.curr = start
         2.while(curr != null)
                  2.1 Print  curr -> info
                   2.2 curr = curr -> next
}
```

```
Algorithm bwddtraversing() {
    1.curr = last
     2.while(curr != null)
              2.1 Print  curr -> info
              2.2 curr = curr -> prev
}
```

: INSERTION

```
Start=NULL
Algorithm InsertAtBEG() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info = data]
3. If (Start == NULL)
          3.1 new1 -> next = NULL
          3.2 new1 -> prev = NULL
          3.3 Last=new1
          3.4 Start=new1

    else
          3.1 new1 -> next = Start
          3.2 Start -> prev = new1
          3.3 new1 -> Prev = NULL
          3.4 Start = new1
}
```

```
Start=NULL
Algorithm InsertAtEnd() {
1. Create node [(new1 = (struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info =data]
3.if(Start == NULL)
          3.1 new1 -> next = NULL
          3.2 new1 -> prev = NULL
          3.3 Last=new1
          3.4 Start=new1
        else
          3.1 Last -> next = new1
          3.2 new1 -> prev = Last
          3.3 new1 -> next = NULL
          3.4 Last = new1
}
```

```
Algorithm InsertAtSpec() {
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)
        4.1 new1 -> next = Start
        4.2 Start -> prev = new1
        4.3 new1 -> Prev = NULL
        4.4 Start = new1
  Else
        4.1 Previous = Start
        4.2 Count = 1
        4.3 While( Count <= Location - 1 && Previous !=NULL)
                4.3.1 Previous = Previous -> next
                4.3.2 Count++
        4.4 new1 -> prev = Previous
        4.5 IF ( Previous -> next == NULL)
                        4.5.1 new1 -> next = NULL;
                        4.5.2 Previous -> next = new1
                        4.5.3 Last = new1
                 Else
                        4.5.1 new1 -> next = Previous -> next
                        4.5.2 Previous -> next -> prev = new1
                        4.5.3 Previous -> next = new1
}
```

: DELETION

```
Algorithm DeleteAtBeg() {
1. If (Start == NULL)
        1.1 Print "underflow“
else
        1.1 Current = Start
        1.2 Start = Start -> next
        1.3 Start -> prev = NULL
        1.3 Current -> next = NULL
        1.4 Current -> prev = NULL
        1.5 Release the memory [ free (Current) ]
}
```

```
Algorithm DeleteAtSpec() {
1. Enter the Location
2. Current = Start
3. Previous = NULL
4. If (Start == 0)
    4.1 Print "underflow“
else If ( Location == 1)
    4.1 Start = Start -> next
    4.2 Start -> prev = NULL
    4.3 Current -> next = NULL
    4.4 Release the memory [ free (Current) ]
else
    4.1 for (i=1; i<Location; i++)
        4.1.1 Previous = Current
        4.1.2 Current = Current -> next
    4.2 if ( Current -> next == NULL)
            4.2.1 Previous -> next = NULL
            4.2.2 Last = Previous
      else
            4.2.1 Previous -> next = Current -> next
            4.2.2 Current -> next -> prev = Previous
4.3 Current -> next = NULL
4.4 Current -> prev = NULL
4.5 Release the memory [ free (Current) ]
}
```

```
Algorithm DeleteAtEnd() {
1. If (Start == NULL)
    1.1 Print "underflow“
else If (Start -> next == NULL )
    1.1 Release the memory [ free (Start) ]
    1.2 Start = NULL
    1.3 Last = NULL
else
    1.1 Current = Last -> prev
    1.3 Current -> next = NULL
    1.4 Last -> prev = NULL
    1.4 Release the memory [ free (Last) ]
    1.5 Last = Current
}
```