UNIT IV
Linked List

Prof. Pujashree Vidap

Contents

- Introduction to Static and Dynamic Memory Allocation
- Linked List: Introduction, of Linked Lists
- Realization of linked list using dynamic memory management
- Linked List as ADT
- Types of Linked List: singly linked, linear and Circular Linked Lists, Doubly Linked List, Doubly Circular Linked List
- Primitive Operations on Linked List-Create, Traverse, Search, Insert, Delete, Sort, Concatenate.
- Polynomial Manipulations-Polynomial addition.
- Generalized Linked List (GLL) concept and Representation of Polynomial using GLL.

Memory Usage

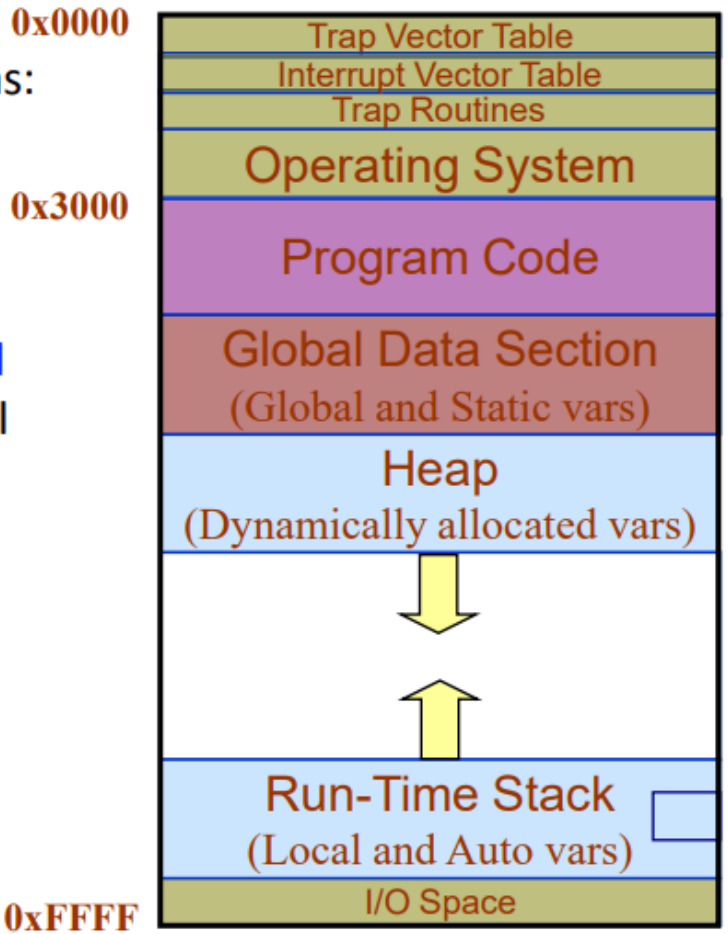
Variable memory is allocated in three areas:

- Global data section
- Run-time stack
- Dynamically allocated - heap

Global variables are allocated in the global data section and are accessible from all parts of the program.

Local variables are allocated during execution on the run-time stack.

Dynamically allocated variables are items created during run-time and are allocated on the heap.



Static Vs Dynamic Memory Allocation

Sr. No.	Static Memory Allocation	Dynamic Memory Allocation
1	It uses stack for managing the static allocation of memory	It uses heap for managing the dynamic allocation of memory
2	There is no memory re-usability	There is memory re-usability and memory can be freed when not required
3	Once the memory is allocated, the memory size can not change.	When memory is allocated the memory size can be changed.
4	We cannot reuse the unused memory.	This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it.
5	Execution is faster than dynamic memory allocation.	Execution is slower than static memory allocation.
6	In this memory is allocated at static time.	In this memory is allocated at run time.
7	In this allocated memory remains from start to end of the program.	In this allocated memory can be released at any time during the program.
8	Example: This static memory allocation is generally used for array.	Example: This dynamic memory allocation is generally used for linked list.

- Problem 1:
- Suppose you have to write an algorithm to generate and store all prime numbers between 1 and 10,00,000 and display them.
- How will you solve this problem?

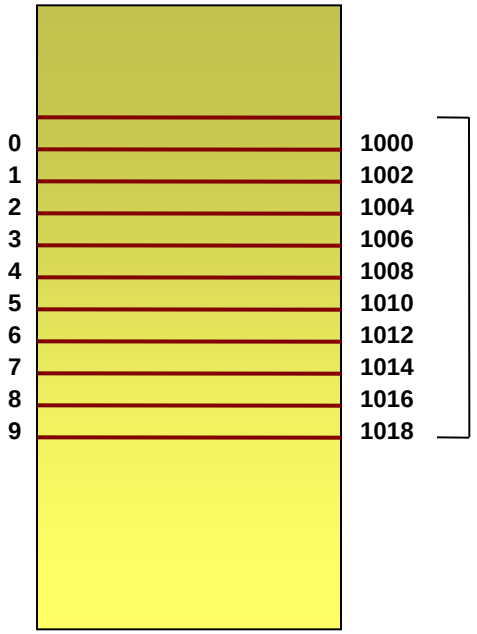
- Problem 2:
- List of three letter words ending in AT
- (BAT, CAT, EAT, FAT, HAT, MAT, OAT, PAT, SAT, VAT, WAT)
- We want to add GAT means gun then we have to move HAT, MAT.....if we have so many insertions in middle then excessive data movement is required. Same is applicable for delete operations.

- Consider the following algorithm, which uses an array to solve this problem:
  1. Set I = 0
  2. Repeat step 3 varying N from 2 to 1000000
  3. If N is a prime number
     a. Set A[I] = N // If N is prime store it in an array
     b. I = I + 1
  4. Repeat step 5 varying J from 0 to I-1
  5. Display A[J] // Display the prime numbers stored in the array

- What is the problem in this algorithm?
  - The number of prime numbers between 1 and 10,00,000 is not known. Since you are using an array to store the prime numbers, you need to declare an array of arbitrarily large size to store the prime numbers.
  - Disadvantages of this approach, suppose you declare an array of size N:
    - If the number of prime numbers between 1 and 10,00,000 is more than N then all the prime numbers cannot be stored.
    - If the number of prime numbers is much less than N, a lot of memory space is wasted.

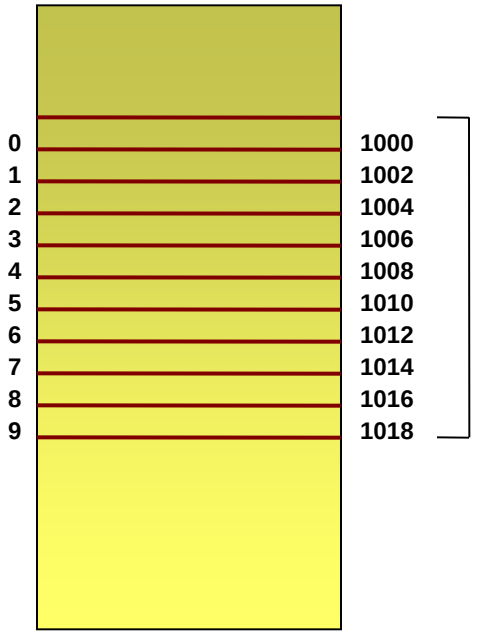
- Thus, you cannot use an array to store a set of elements if you do not know the total number of elements in advance.
- How do you solve this problem?
  - By having some way in which you can allocate memory as and when it is required.

- If you know the address of the first element in the array you can calculate the address of any other elements as shown:
  - Address of the first element + (size of the element × index of the element)
    - When you declare an array, a contiguous block of memory is allocated.
    - Let us suppose you declare an array of size 10 to store first 10 prime numbers.



Memory representation

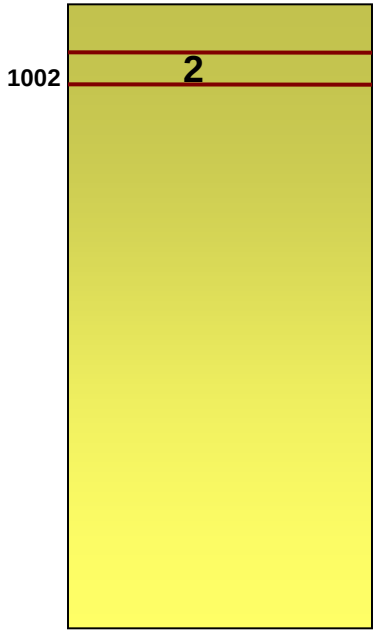
- When memory is allocated dynamically, a block of memory is assigned arbitrarily from any location in the memory.
- Therefore, unlike arrays, these blocks may not be contiguous and may be spread randomly in the memory.



Memory representation

- Let us see how this happens by allocating memory dynamically for the prime numbers.

Allocate memory for 2

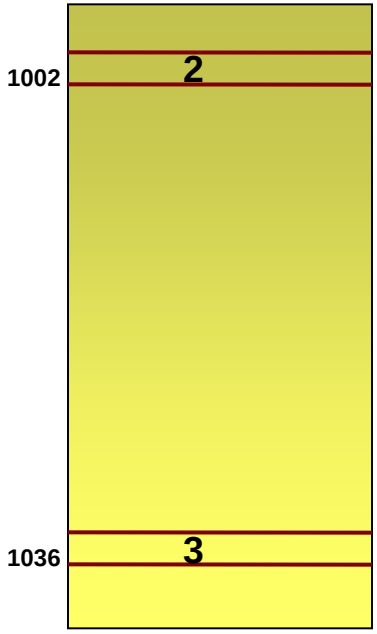


Memory allocated for 2

Memory representation

- Let us see how this happens.

Allocate memory for 3

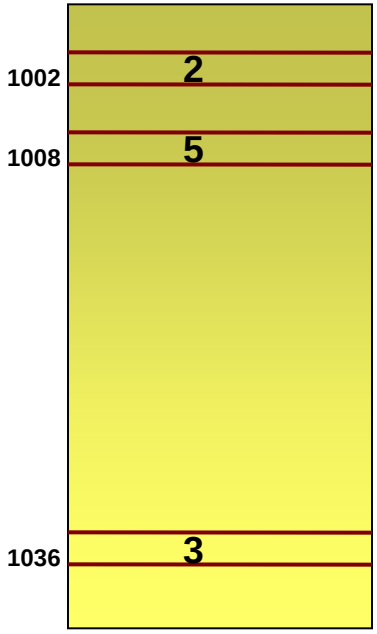


Memory allocated for 3

Memory representation

- Let us see how this happens.

Allocate memory for 5

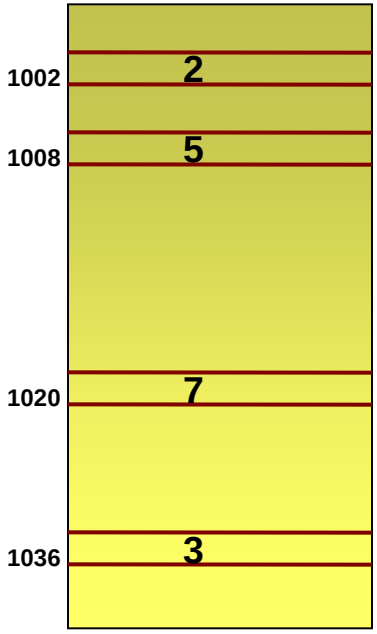


Memory allocated for 5

Memory representation

- Let us see how this happens.

Allocate memory for 7

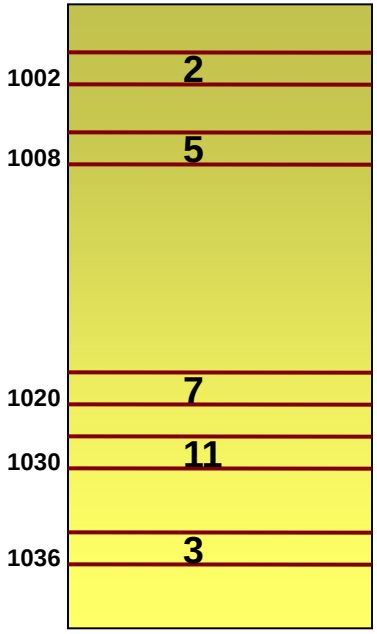


Memory representation

Memory allocated for 7

- Let us see how this happens.

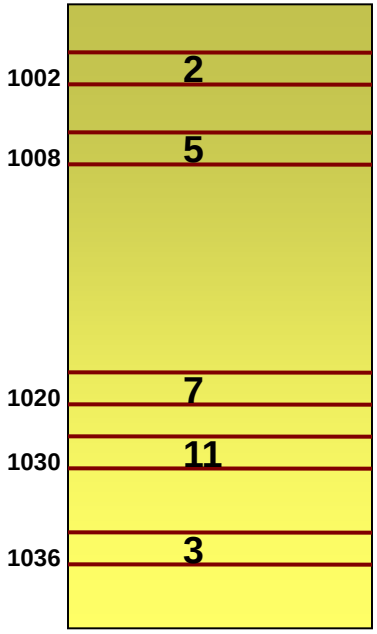
Allocate memory for 11



Memory representation

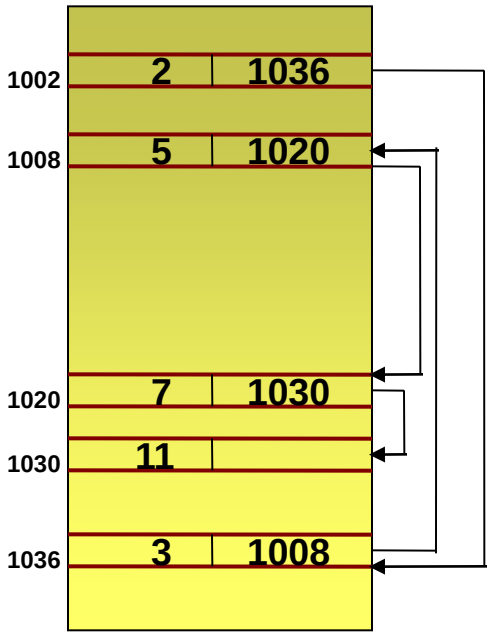
Memory allocated for 11

- To access any element, you need to know its address.
- Now, if you know the address of the first element, you cannot calculate the address of the rest of the elements.
- This is because, all the elements are spread at random locations in the memory.



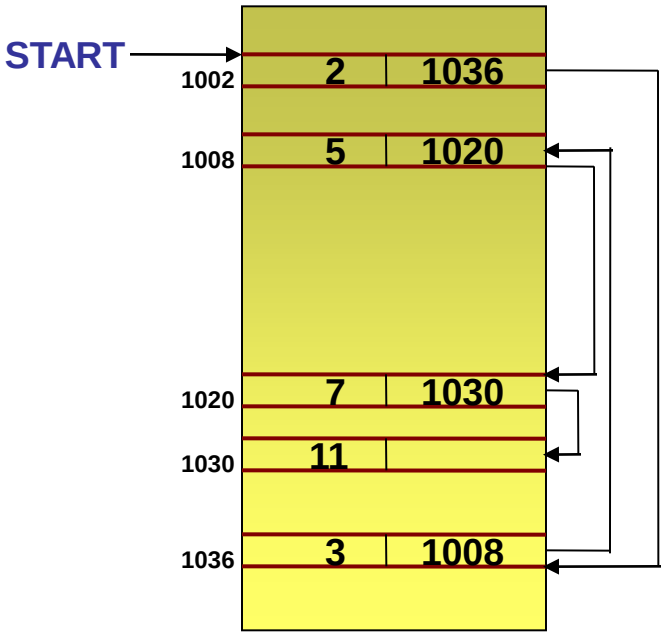
Memory representation

- Therefore, it would be good if every allocated block of memory contains the address of the next block in sequence.
- This gives the list a linked structure where each block is linked to the next block in sequence.



Memory representation

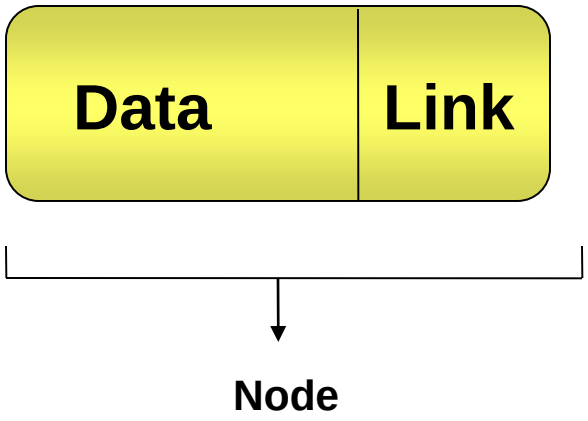
- An example of a data structure that implements this concept is a linked list.
- You can declare a variable, START, that stores the address of the first block.
- You can now begin at START and move through the list by following the links.



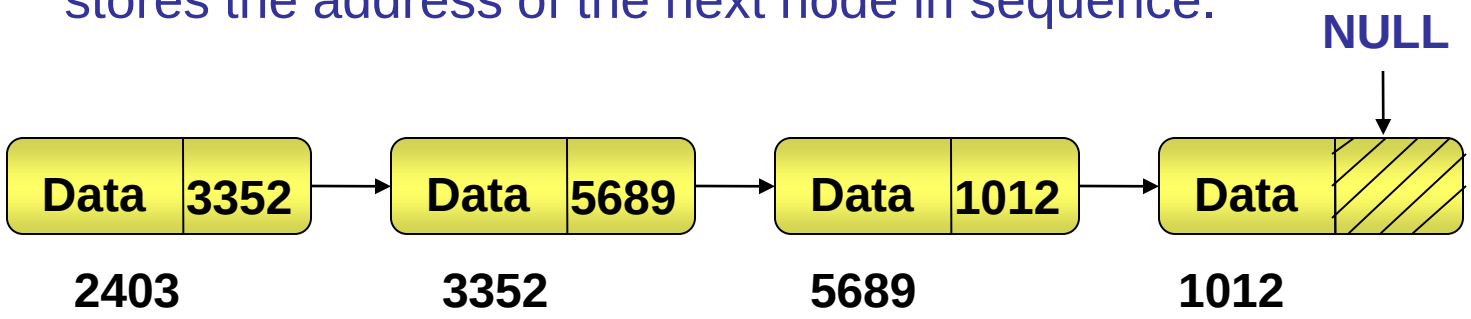
Memory representation

- Realization of Linked list:
  - Is a dynamic data structure.
  - Allows memory to be allocated as and when it is required.
  - Items/elements are placed anywhere in memory
  - Consists of a chain of elements, in which each element is referred to as a node.

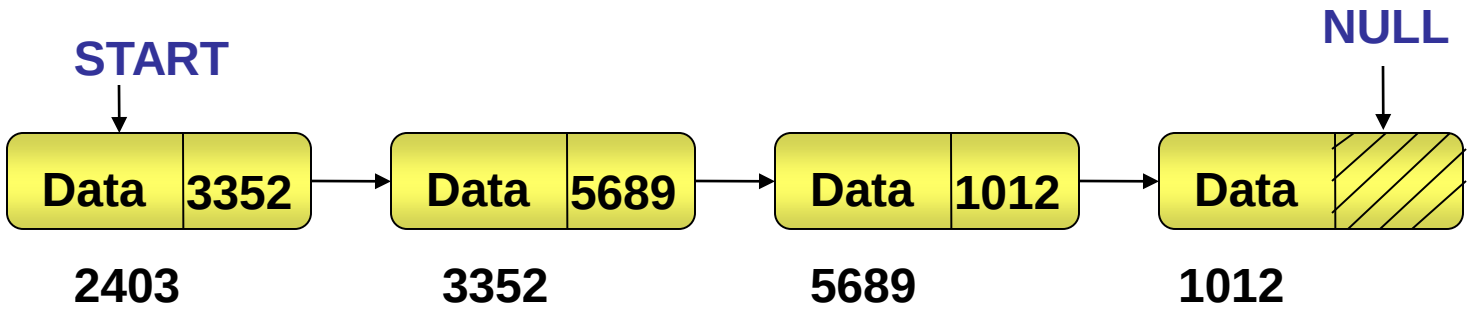
- A node is the basic building block of a linked list.
- A node consists of two parts:
  - **Data**: Refers to the information held by the node
  - **Link**: Holds the address of the next node in the list



- The last node in a linked list does not point to any other node. Therefore, it points to NULL.
  - All the nodes in a linked list are present at arbitrary memory locations.
  - Therefore, every node in a linked list has link field that stores the address of the next node in sequence.



- To keep track of the first node, declare a variable/pointer, START, which always points to the first node.
- When the list is empty, START contains null.



HOW TO CREATE A NODE IN C

```
struct node
{
int  info;
struct node  *next;
};
struct node *new1;
new1=(struct node*) malloc(sizeof(struct node))
new1->info=3;
new1->next=NULL;
```

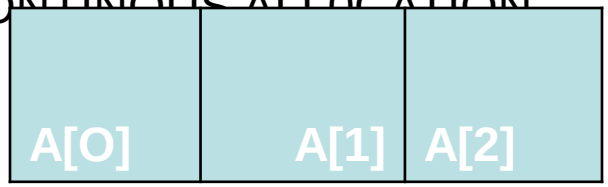
HOW TO CREATE A NODE

```
class node
{
int  info;
node  *next;
};
node *new1;
new1=new new1
new1->info=3;
new1->next=NULL;
```

DIFFERENCE

- SEQUENTAL ORGANISATION

1)CONTINOUS ALLOCATION



2)STATIC MEMORY ALLOCATION.

3)DELETION IS LOGICAL NOT ACTUAL.

4)RANDOM ACCESS

5)FROM THE BASE ADDRESS, ADDRESS OF EVERY VARIABLE CAN BE CALCULATED.

6)INSERTION ,DELETION IS DIFFICULT

LINKED ORGANISATION

1)ELEMENTS ARE ALLOCATED NON –CONTIGUOSLY

2)DYNAMIC MEMORY ALLOCATION SO NO WASTE.

3)ACTUAL DELETION IS POSSIBLE.

4)SEQUENTIAL ACCESS.

5)EACH NODE STORE THE ADDRESS OF NEXT NODE

.

6)INSERTION ,DELETON IS EASY.

PRIMITIVE OPERATIONS ON LINKLIST

- CREATION
- INSERTION
- DELETION
- TRAVERSING

Realization of link list using arrays

- We can represent link list using array

```
struct node
{
  int data;
  int next
 } A[10];
```

Consider list={1,2,3,4,5}

	DATA	NEXT
A[0]	2	3
A[1]	1	0
A[2]	5	-1
A[3]	3	4
A[4]	4	2

Insertion and deletion

Data	next
-1	0
-1	0
1	4
-1	0
3	5
4	-1

Suppose we have to insert after 1 and position of 1 is 2,
1)Find the free space suppose j
2)A[j].next=A[2].next
3)A[2].next=j

Delete any element

Data	next
-1	0
-1	0
1	4
-1	0
3	5
4	-1

i →

j →

1)Let i is the position of previous element and j is the position of element to be deleted

2)a[i].next=a[j].next

3)a[j].data =-1

a[j].next =0

Display elements

```
1) j=START
while(j!=-1) {
 print (a[j].data)
  j=a[j].next
}
```

Data	next
-1	0
-1	0
1	4
-1	0
3	5
4	-1

Overview of static memory management

In static memory allocation the amount of memory to be allocated is decided during compile time itself.

This is known as static memory allocation.

Limitations:

- **It is rigid. The programmers are forced to predict the total amount of data the program will utilize.**
- **During run time, if more memory is required, static memory allocation cannot fulfil the need.**
- **Once a certain memory block is no longer of any use to the program, memory allocated to it cannot be released immediately.**
- **The memory will continue to be held up until the end of the block in which the variable was created.**

Dynamic memory management

Solution: Dynamic memory management overcomes the drawbacks of static memory allocation.

- **Although memory gets allocated and deallocated during run time only, the decisions to do so can be taken dynamically in response to the requirements arising during run time itself.**

- **Once a certain block of memory is no longer required, it can immediately be returned to the OS.**

- **The OS can then allocate the deallocated memory block if the need arises.**

- **For this, code utilizing the relevant functions and operators provided by C and C++ has to be explicitly written in the source code.**

NEW AND DELETE OPERATOR IN C++

- **Dynamically allocates memory on the heap.**
- **In the C++ programming language, new is a language construct that dynamically allocates memory on the heap .**
- **New attempts to allocate enough memory on the heap for the new data. If successful, it initializes the memory and returns the address to the newly allocated and initialized memory.**
- **However if new cannot allocate memory on the heap it returns null or raise an exception.**
- **A call to delete, which calls the destructor and returns the memory allocated by new back to the heap, must be made for every call to new .**

Allocating memory dynamically

- Syntax: p_var = new typename;

  int *p = new int;

- allocates an integer, set to 5.

  int *p = new int(5);

- int *p= new int[5];

Releasing dynamically allocated memory

```
int *p_var = new int;
  delete p_var;
 int *p_array = new int[50];
delete[ ] p_array;
```

Example

```
#include <iostream.h>
int main ()
{ int i,n;
 int * p;
cout << "How many numbers would you like to type? ";
 cin >> i;
 p= new int[i];
if (p == null)
cout << "Error: memory could not be allocated";
•
else {
for (n=0; n<i; n++)
 { cout << "Enter number";
 cin >> p[n];
}
cout << "You have entered: ";
 for (n=0; n<i; n++)
 cout << p[n] << ", ";
 delete[] p;
 } }
```

Dynamic memory allocation in c

To allocate memory use

```
void *malloc(size_t size);
```

- Takes number of bytes to allocate as argument.
- Use sizeof to determine the size of a type.
- Returns pointer of type void *. A void pointer may be assigned to any pointer.
- If no memory available, returns NULL.

```
int *ip = (int*)malloc(100 *sizeof(int))
```

calloc()

```
void *calloc(size_t nitem, size_t size);
```

- The main difference is that the values stored in the allocated memory space are zero by default in calloc(). With malloc(), the allocated memory could have any value.
- calloc() requires two arguments - the number of variables you'd like to allocate memory for and the size of each variable.

```
(float*)calloc(100, sizeof(float));
```

realloc()

- If you find you did not allocate enough space use realloc().

```
int *ip;

ip = (int*)malloc(100 * sizeof(int));
If(ip==NULL){"CANNOT ALLOCATE");
ip = (int*)realloc(ip, 200 * sizeof(int));
```

- If additional space is not available in the same region ,it will create in new region and move the content of old to new block.
- **If block cannot be allocated or new size is 0,realloc returns null pointer and original block is lost(freed).**

free()

To release allocated memory use

```
free()
```

- Deallocates memory allocated by malloc().
- Takes a pointer as an argument.

e.g.

```
free(newPtr);
```

Difference between malloc and new

- Sizeof operator not required in new.
- No need to typecast
- New and delete can be overloaded so u can customize allocating system.
- New is typesafe where malloc is not
- For new , the constructor is called

Ex. foo *f2 = (foo*)(malloc(sizeof(foo)));

foo *f3 = (foo*)(malloc(1)); // No error but bad

NOTE:

- Not to use new and malloc in same program
- No guarantee that they are mutually compatible.

Dynamic Memory Deallocation

- **A block of memory allocated dynamically can be deallocated dynamically. Once it is not in use any more, a dynamically allocated block of memory should definitely be returned to the OS.**
- **If they have not been returned to the OS, they will remain locked up. This is known as a memory leak. If more memory is required, the OS will not allocate this block of memory.**

Ex void f1() {

int *ptr = (int *) malloc(sizeof(int));

/* Do some work */

return; /* Return without freeing ptr*/ }

A memory leak is memory which hasn't been freed, there is no way to access (or free it) now, as there are no ways to get to it anymore.

Dynamic Memory Deallocation

- **Myth: A misconception about the *delete* operator is due to the commonly used phrase 'deleting the pointer'. One may think that the memory being occupied by the pointer itself gets removed if the *delete* operator is used on the pointer.**

  Reality: When the delete operator is used on a pointer, the pointer continues to occupy its own block of memory and continues to have the same value that is the address of the first byte of the block of memory that has just got deallocated. Thus, the pointer continues to point at the same block of memory(dangling pointer). This will lead to run-time errors if the pointer is dereferenced.

PROBLEMS

- Memory leak
- Dangling pointers
- Problem of memory allocation failure

Link list ADT

class LinkedList

{ //collection of nodes and each node is having data and
  //address to next node

public:

create(); //construct link list.

insert();//using this function a node can be inserted at any specific position.

deleteNode(); //using this function a node can be deleted from specific position.

search();// using this function specific node can be searched.

traversal();

}

Inserting a Node in a Singly-Linked List (Contd.)

- A new node can be inserted at any of the following positions in the list:
  - Beginning of the list
  - End of the List
  - At specific position
  - In a Sorted Linked List

Inserting a Node a Singly-Linked LIST

- Refer to the given algorithm to insert a node at the end of the linked list.

1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):

   1. START=new;
   2. LAST=new;
   3. Goto 6

4. Make the next field of LAST point to the new node.

5.Mark the new node as LAST.

6. Make the next field of the new node point to NULL.

- Consider that the list is initially empty.
- START = NULL
- LAST = NULL
- Insert a prime number 2.

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

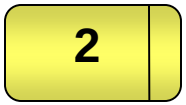
Inserting a Node in a Singly-Linked List (Contd.)

- START = NULL
- Insert a prime number 2.

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)

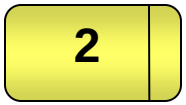
- START = NULL
- LAST = NULL



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)

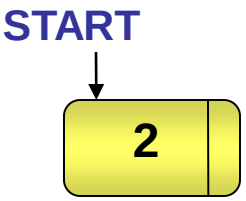
- START = NULL
- LAST = NULL



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

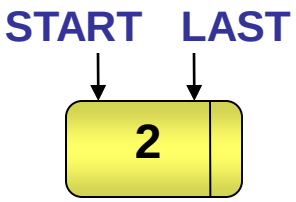
Inserting a Node in a Singly-Linked List (Contd.)

- START = NULL
- LAST = NULL

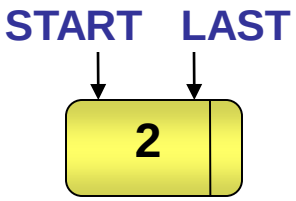


1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

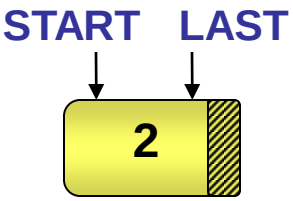
- LAST = NULL



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.



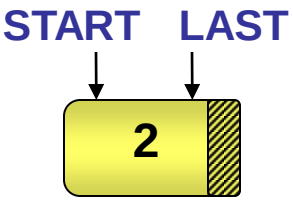
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.



Insertion complete

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

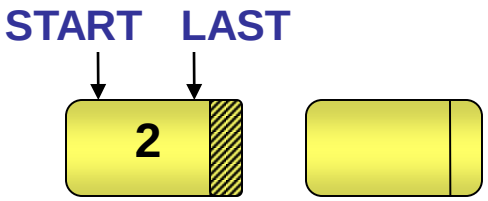
- Insert a prime number 3.



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

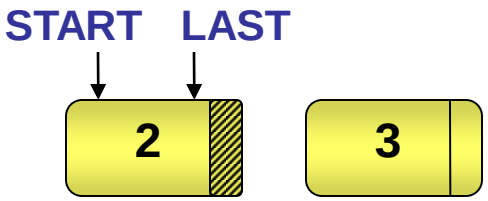
Inserting a Node in a Singly-Linked List (Contd.)

- Insert a prime number 3.



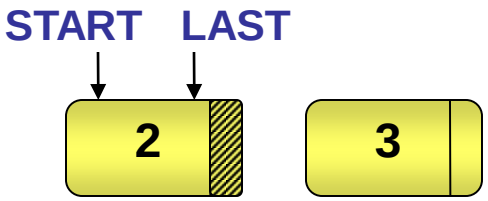
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
   1. Make START point to the new node.
   2. Make LAST point to the new node.
   3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)



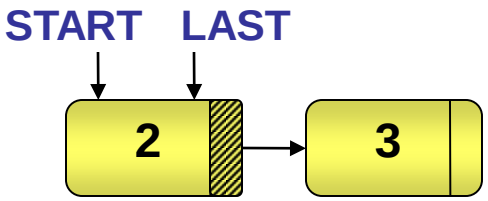
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)



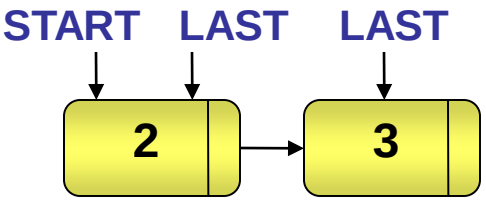
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)



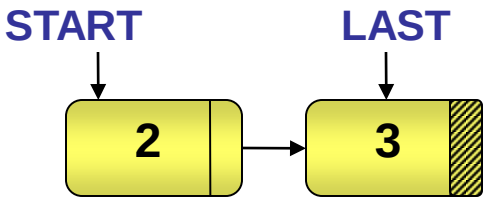
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

Inserting a Node in a Singly-Linked List (Contd.)



Insertion complete

1. Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):
   1. Make START point to the new node.
   2. Make LAST point to the new node.
   3. Go to step 6.

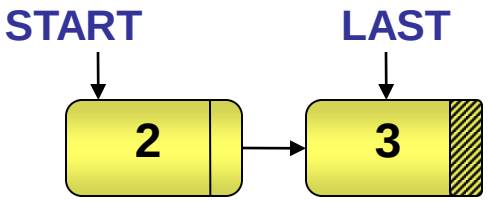
4. Make the next field of LAST point to the new node.

5. Mark the new node as LAST.

6. Make the next field of the new node point to NULL.

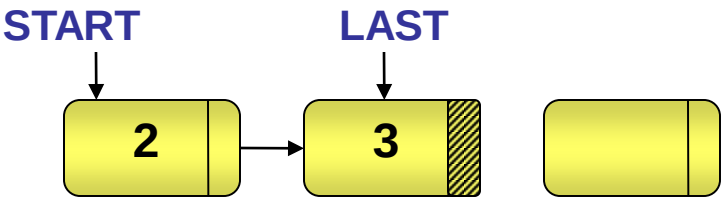
Inserting a Node in a Singly-Linked List (Contd.)

- Insert a prime number 5.

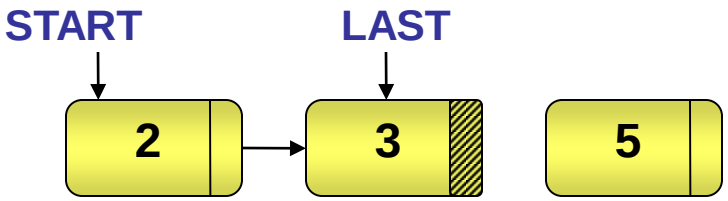


1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.

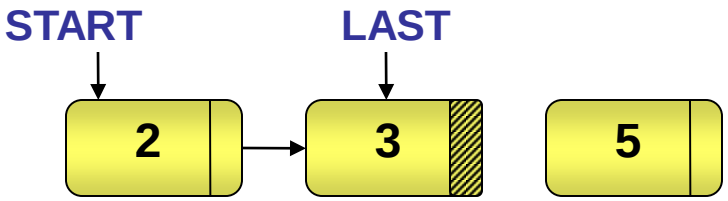
- Insert a prime number 5.



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.
4. Make the next field of LAST point to the new node.
5. Mark the new node as LAST.
6. Make the next field of the new node point to NULL.



1. Allocate memory for the new node.

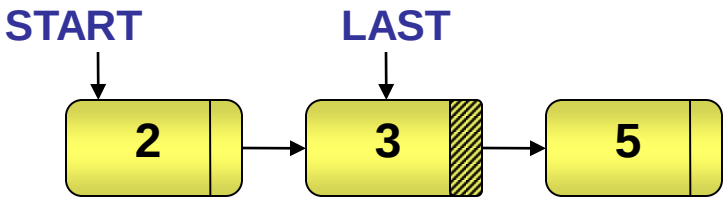
2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):
   1. Make START point to the new node.
   2. Make LAST point to the new node.
   3. Go to step 6.

4. Make the next field of LAST point to the new node.

5. Mark the new node as LAST.

6. Make the next field of the new node point to NULL.



1. Allocate memory for the new node.

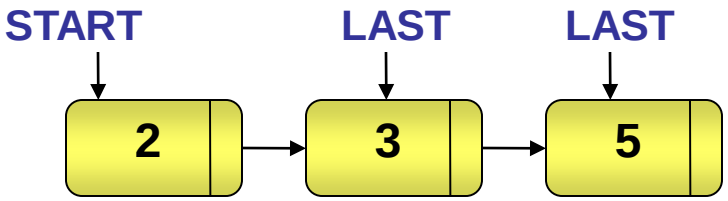
2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):
   1. Make START point to the new node.
   2. Make LAST point to the new node.
   3. Go to step 6.

4. Make the next field of LAST point to the new node.

5. Mark the new node as LAST.

6. Make the next field of the new node point to NULL.



1. Allocate memory for the new node.

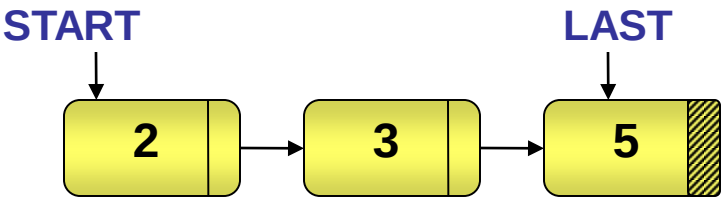
2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):
    1. Make START point to the new node.
    2. Make LAST point to the new node.
    3. Go to step 6.

4. Make the next field of LAST point to the new node.

5. Mark the new node as LAST.

6. Make the next field of the new node point to NULL.



Insertion complete

1. Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. If START is NULL, then (If the list is empty):
   1. Make START point to the new node.
   2. Make LAST point to the new node.
   3. Go to step 6.

4. Make the next field of LAST point to the new node.

5. Mark the new node as LAST.

6. Make the next field of the new node point to NULL.

ALGORITHM TO INSERT NODE AT END

Start=NULL

Algorithm InsertAtEnd()

{

1.Create node new1

// new1 = (struct node*) malloc(sizeof(struct node)) //or

// new1 =new node

2. Enter data [new1 -> info =data]

3.if(Start == NULL)

3.1 Last=new1

3.2 Start=new1

else

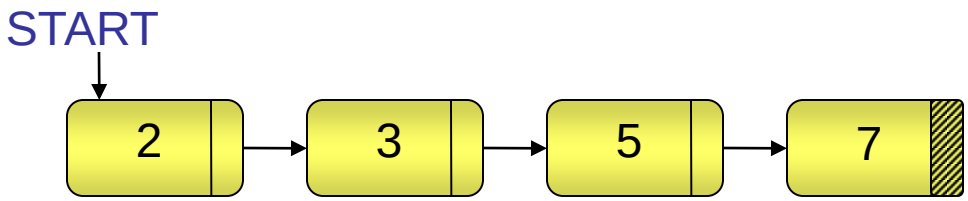
3.1 Last -> next = new1

3.2 Last = new1

4. Last ->next = NULL

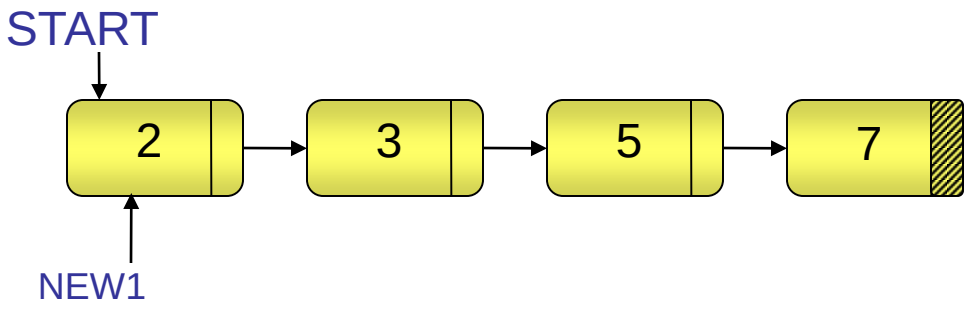
}

- Algorithm for traversing a linked list.
- Write an algorithm to traverse a singly-linked list.



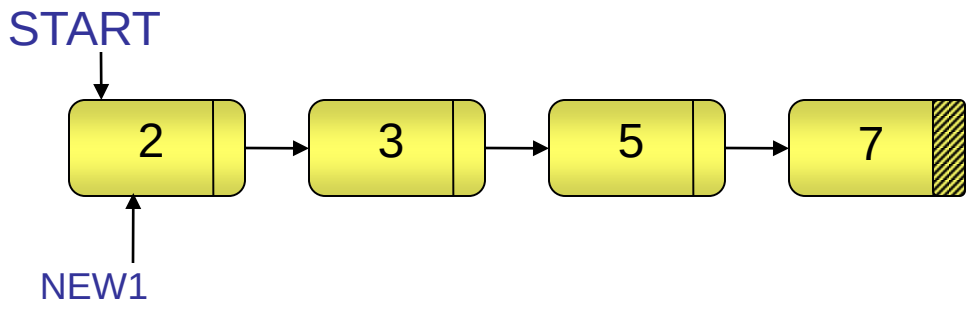
1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.

- Refer to the algorithm to display the elements in the linked list.

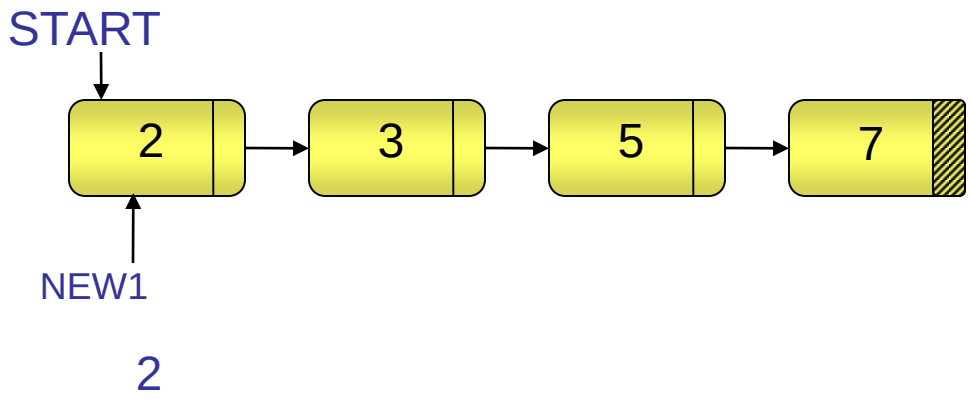


1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.

- Refer to the algorithm to display the elements in the linked list.

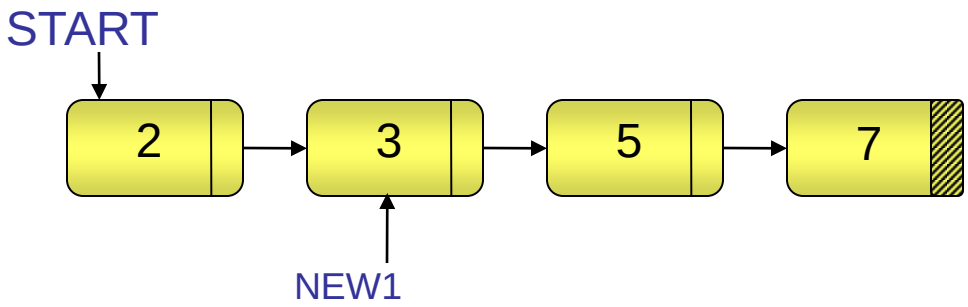


1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



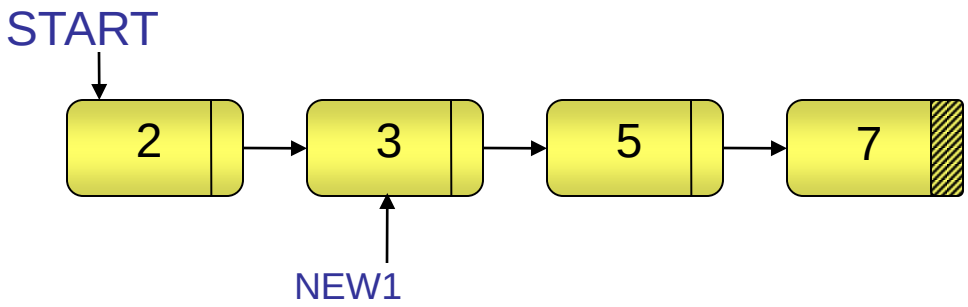
2

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



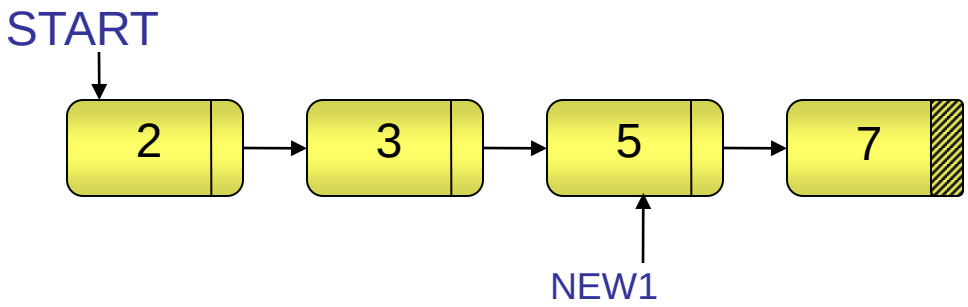
2

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



2 3

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



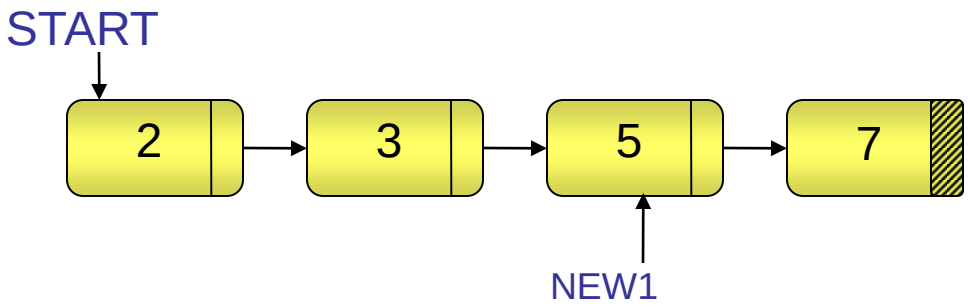
2 3

1. Make NEW1 point to the first node in the list.

1. Repeat step 3 and 4 until NEW1 becomes NULL.

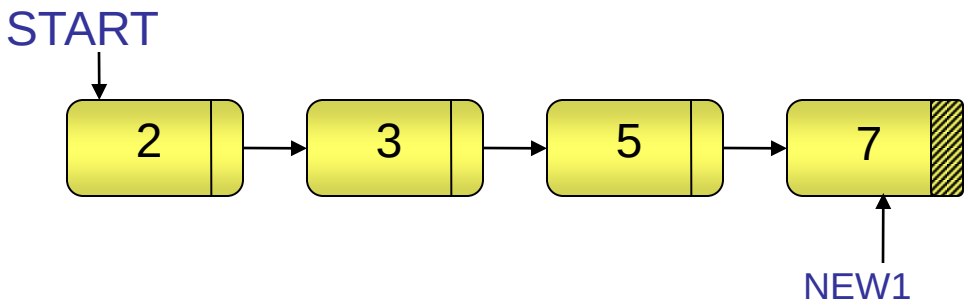
1. Display the information contained in the node marked as NEW1

1. Make NEW1 point to the next node in sequence.



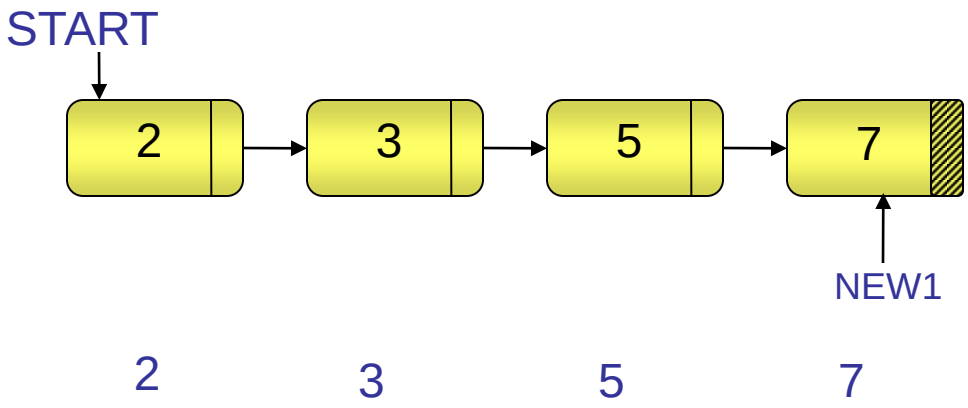
2 3 5

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.

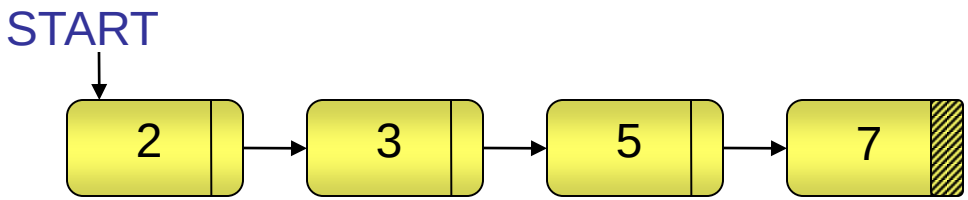


2 3 5

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.



2 3 5 7

1. Make NEW1 point to the first node in the list.
2. Repeat step 3 and 4 until NEW1 becomes NULL.
3. Display the information contained in the node marked as NEW1
4. Make NEW1 point to the next node in sequence.

NEW1 = NULL

Traversal complete

TRAVERSING IN LIST

Algorithm traversing()

{

1.new1 = Start

2.while(new1 != NULL)

    2.1 Print new1 -> info

    2.2 new1 = new1 -> next

}

- Write an algorithm to insert a node in the beginning of a linked list.

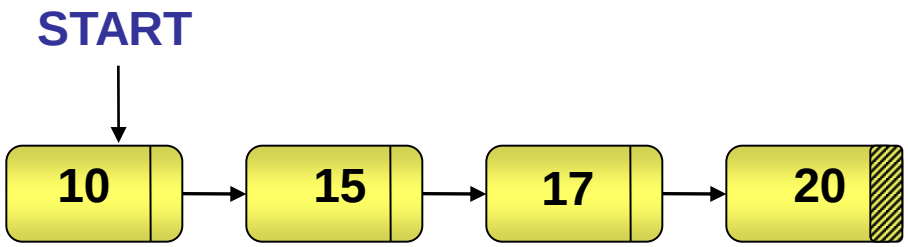
- Algorithm to insert a node in the beginning of a linked list

1.Allocate memory for the new node.

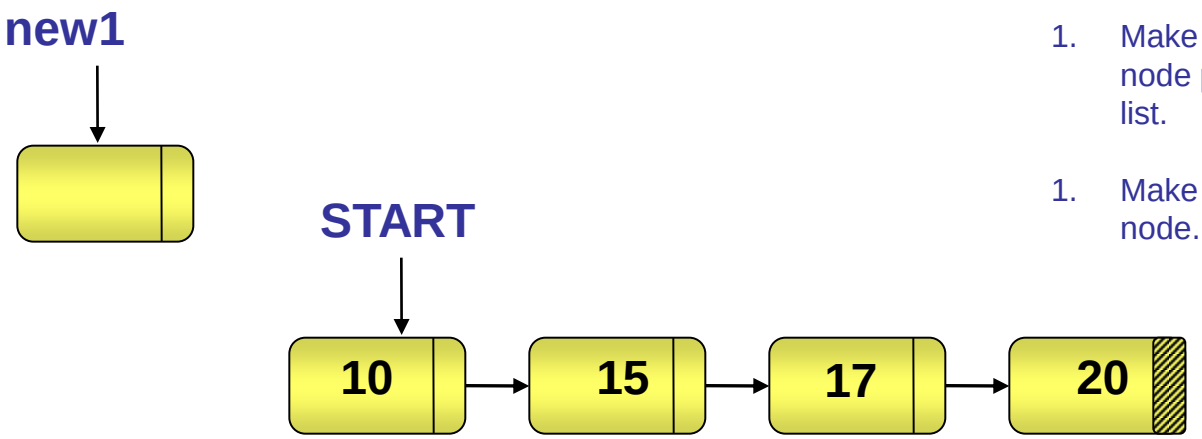
1.Assign value to the data field of the new node.

1.Make the next field of the new node point to the first node in the list.

1.Make START, point to the new node.

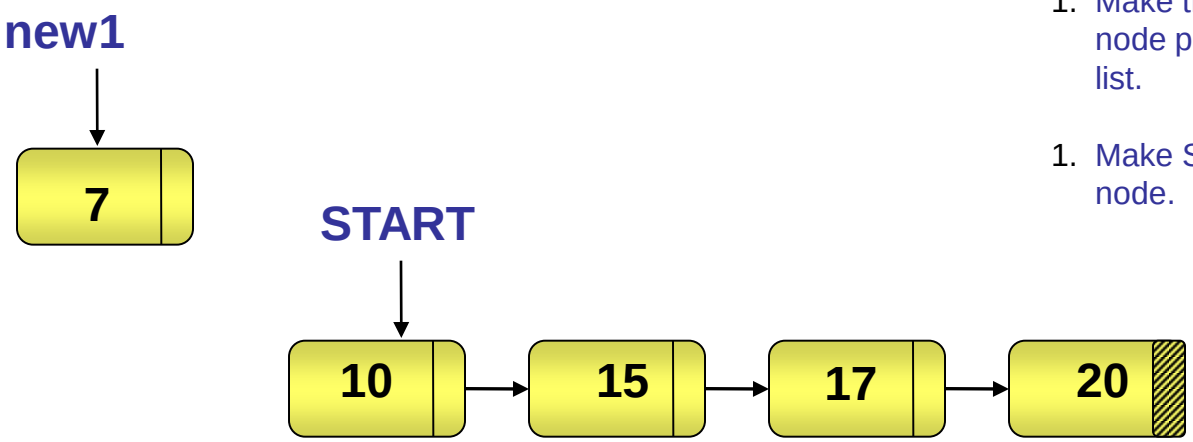


Insert 7



1. Allocate memory for the new node.
1. Assign value to the data field of the new node.
1. Make the next field of the new node point to the first node in the list.
1. Make START, point to the new node.

Insert 7



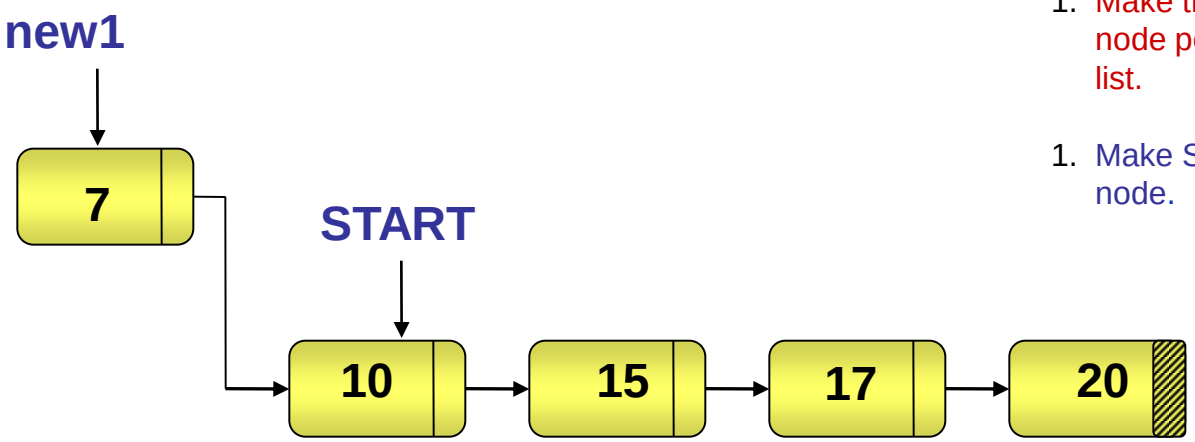
1. Allocate memory for the new node.
1. Assign value to the data field of the new node.
1. Make the next field of the new node point to the first node in the list.
1. Make START, point to the new node.

1. Allocate memory for the new node.

1. Assign value to the data field of the new node.

1. Make the next field of the new node point to the first node in the list.

1. Make START, point to the new node.



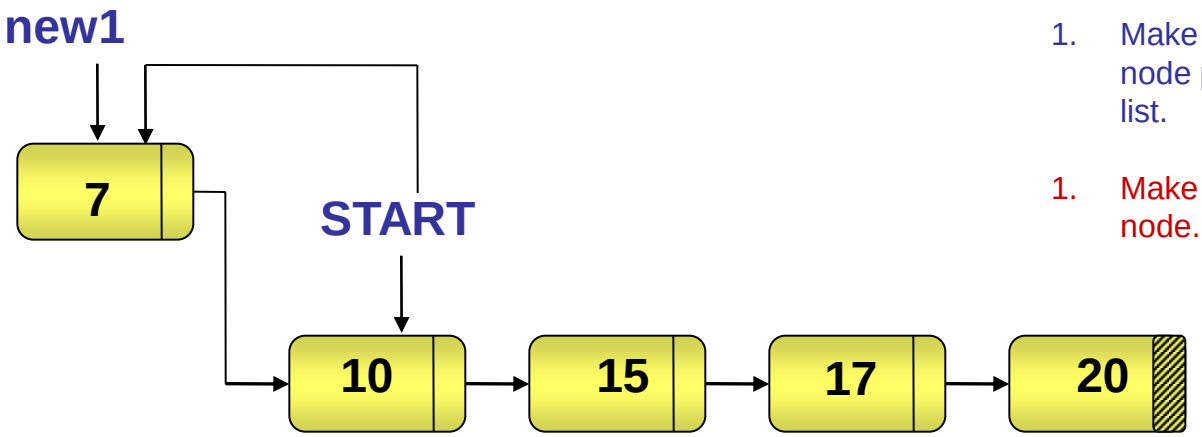
new1 -> next = START

1. Allocate memory for the new node.

1. Assign value to the data field of the new node.

1. Make the next field of the new node point to the first node in the list.

1. Make START, point to the new node.

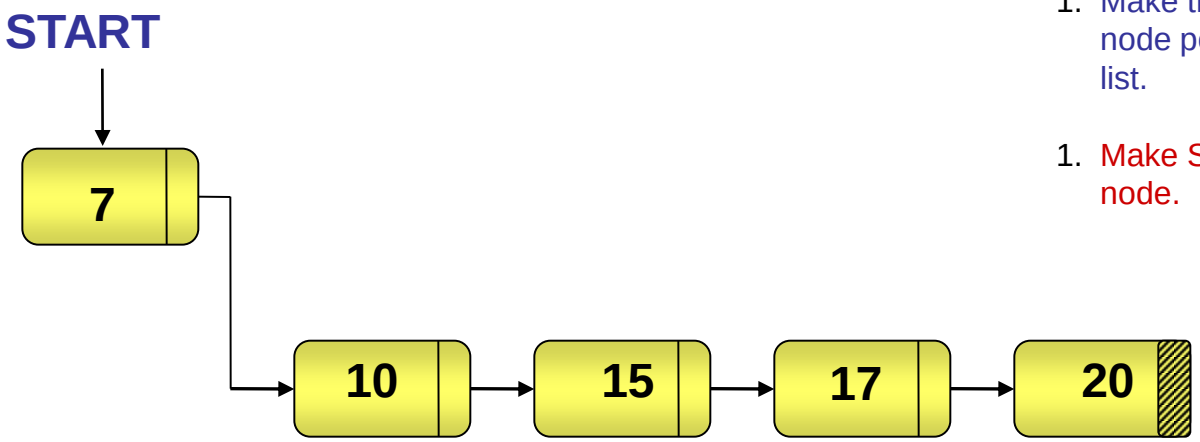


Insertion complete

New1 -> next = START

START = new1

1. Allocate memory for the new node.
1. Assign value to the data field of the new node.
1. Make the next field of the new node point to the first node in the list.
1. Make START, point to the new node.



Insertion complete

New1 -> next = START

START = new1

ALGORITHM TO INSERT NODE AT BEGINING

Start=NULL

Algorithm InsertAtBEG()

{

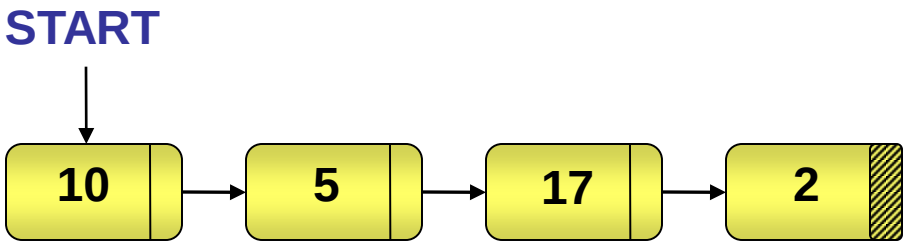
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter data [new1 -> info = data]
3. If (Start == NULL)
    - 3.1 new1 -> next = NULL
    - 3.2 Last=new1
    - 3.3 Start=new1

   else
    - 3.1 new1 -> next=Start
    - 3.2 Start = new1

}

- Write an algorithm to insert a node at the particular position in a linked list.

Insert 16 At LOC = 3



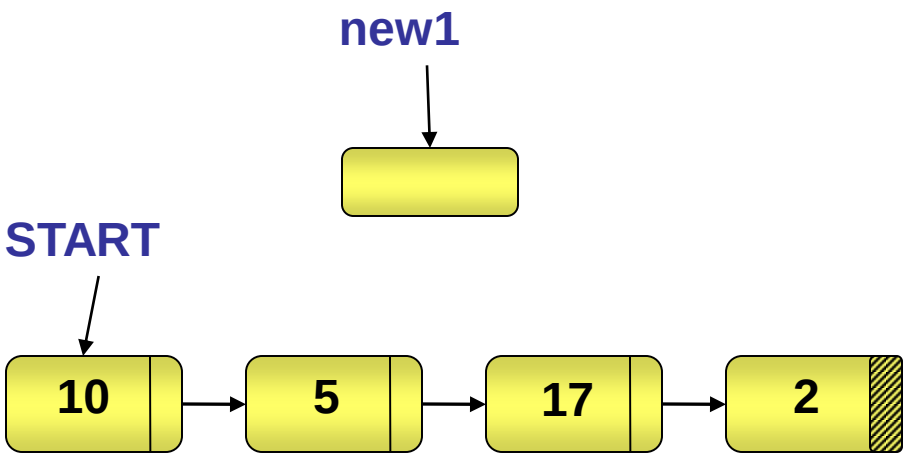
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



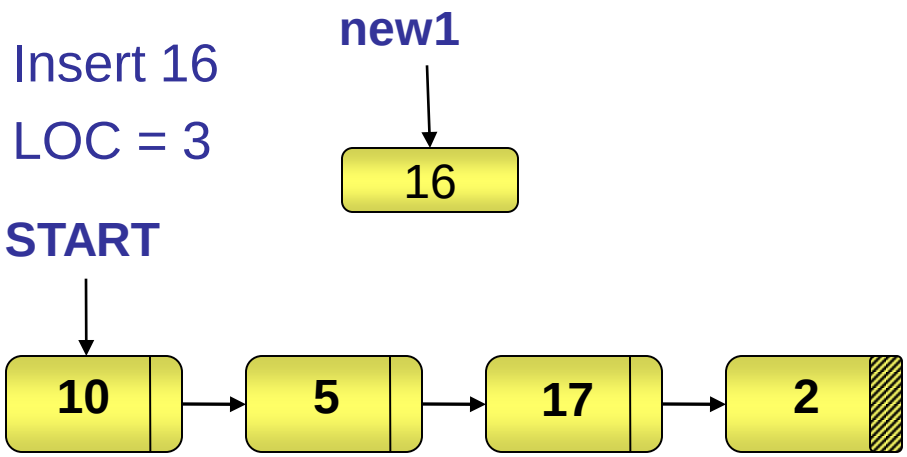
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



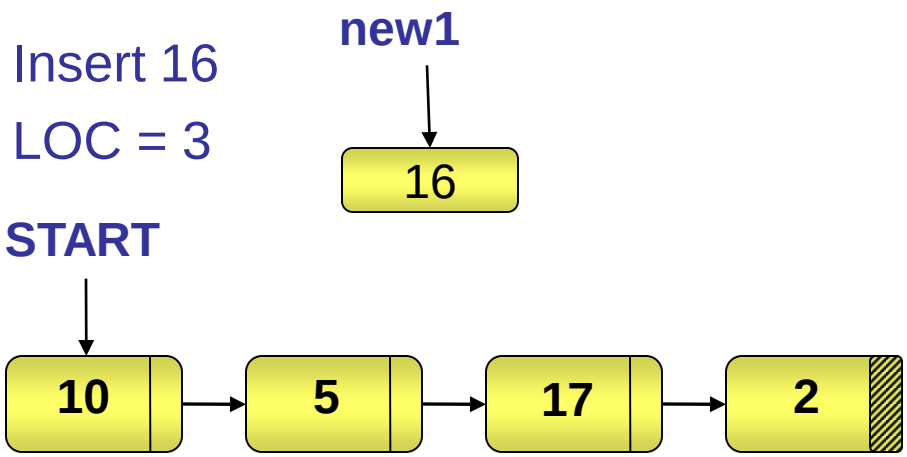
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



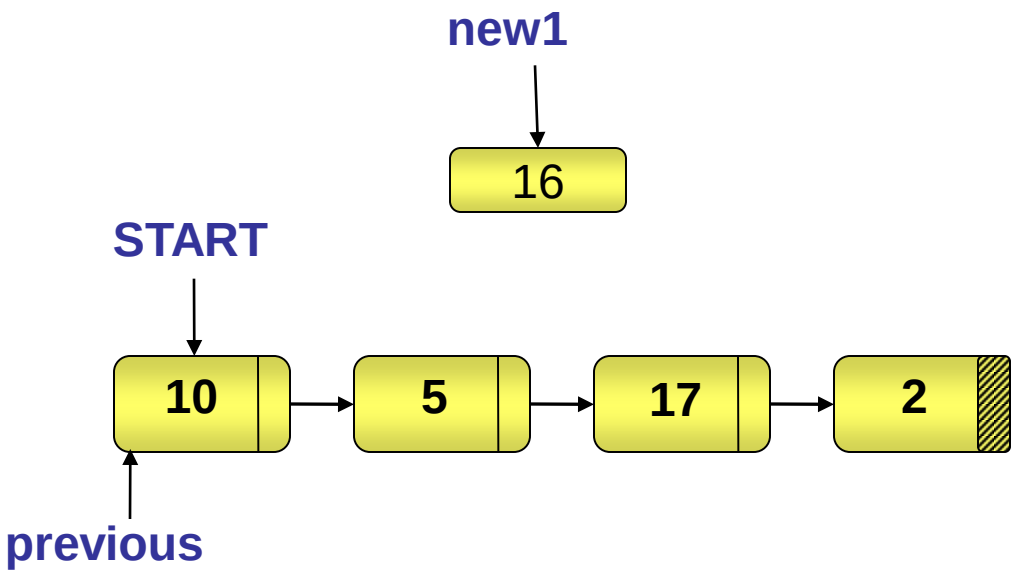
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



Count=1

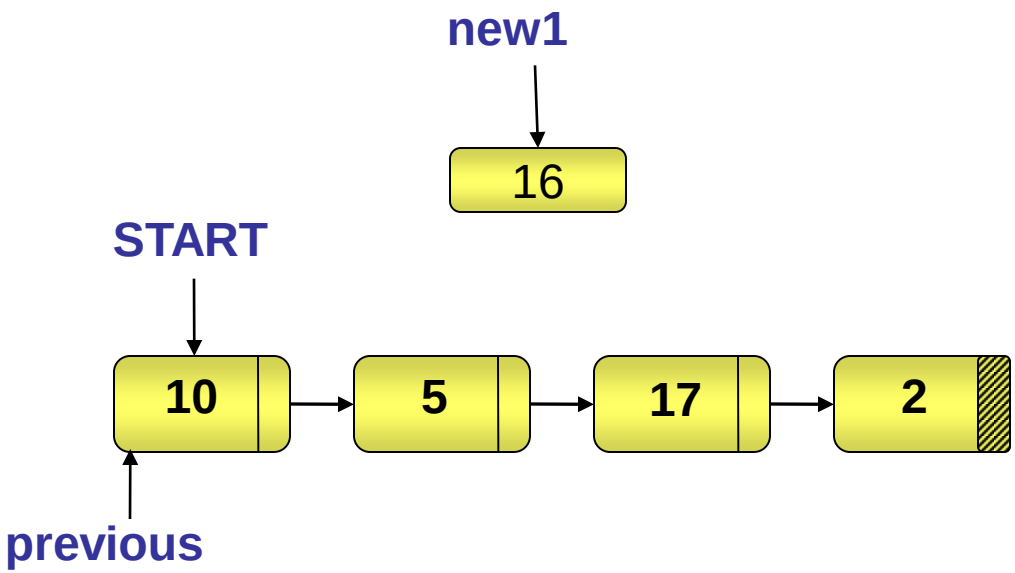
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



Count=1

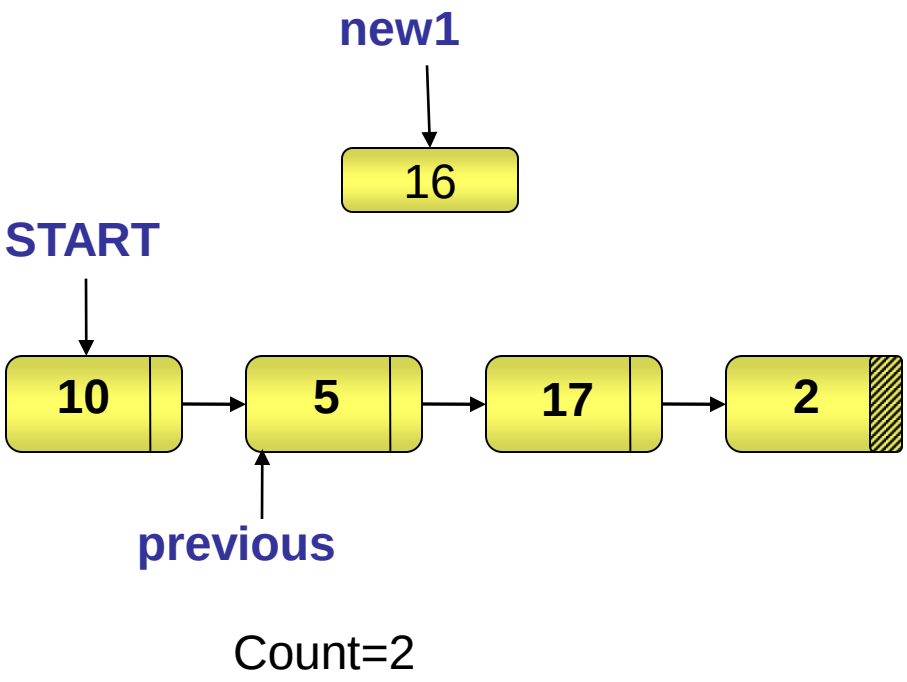
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



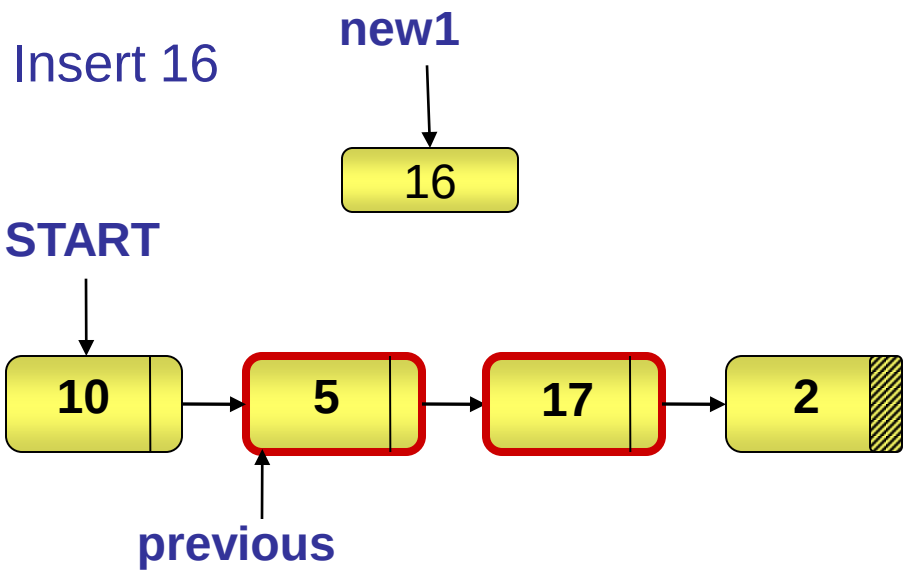
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



Nodes located

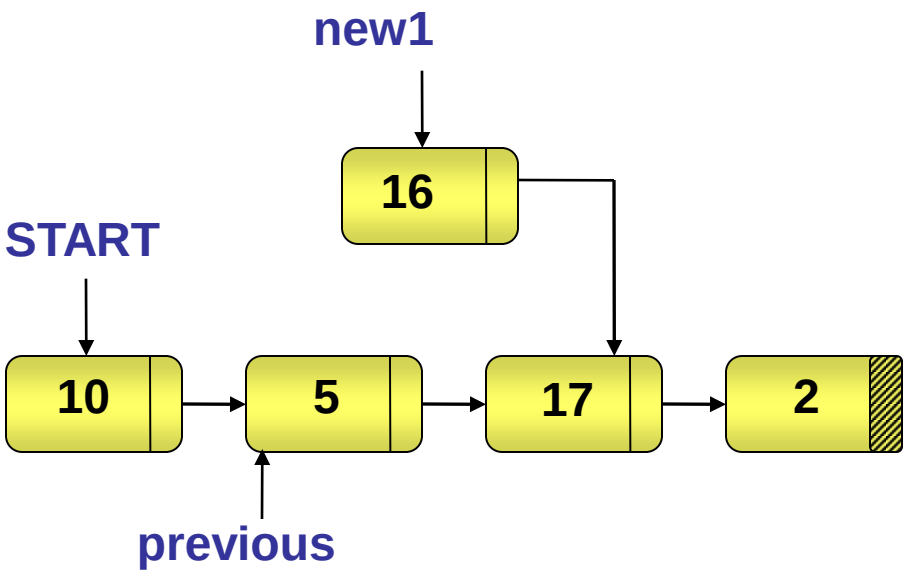
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



new1 -> next = previous->next

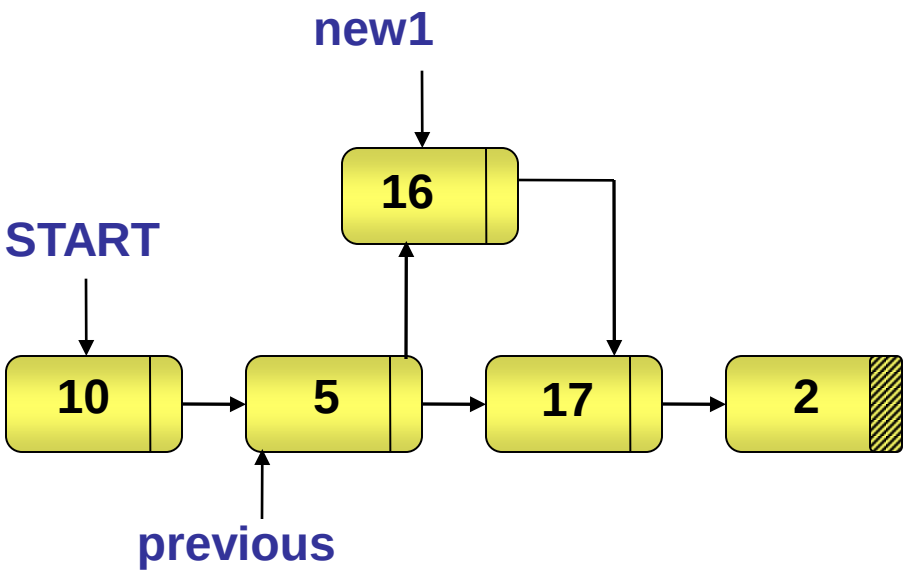
1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.



1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

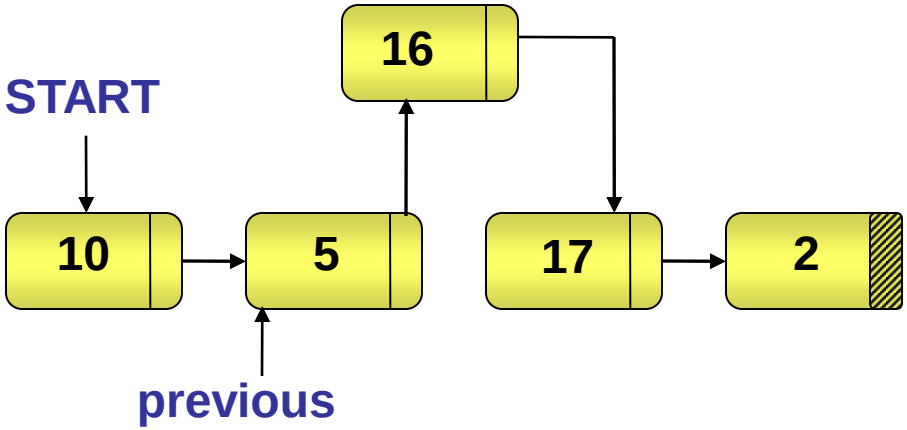
3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.

new1 -> next = previous->next

previous -> next = new1



1.Allocate memory for the new node.

2. Assign value to the data field of the new node.

3. Identify the nodes after which the new node is to be inserted. Mark it as previous
   a. Make previous node point to the first node and set count=1
   b. Repeat step c and step d until count becomes equal to location-1
   c. Count=count+1.
   d. Make previous point to next node in sequence

4. Make the next field of the new node point to the next of previous node

5.Make the next field of previous point to the new node.

new1 -> next = previous->next

previous -> next = new1

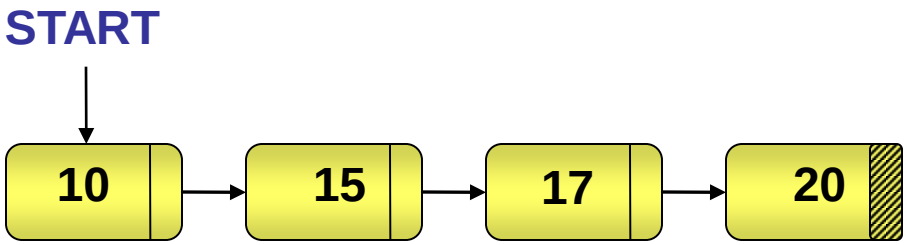
Insertion complete

ALGORITHM TO INSERT NODE At PARTICULAR POSITION IN A LINKED LIST

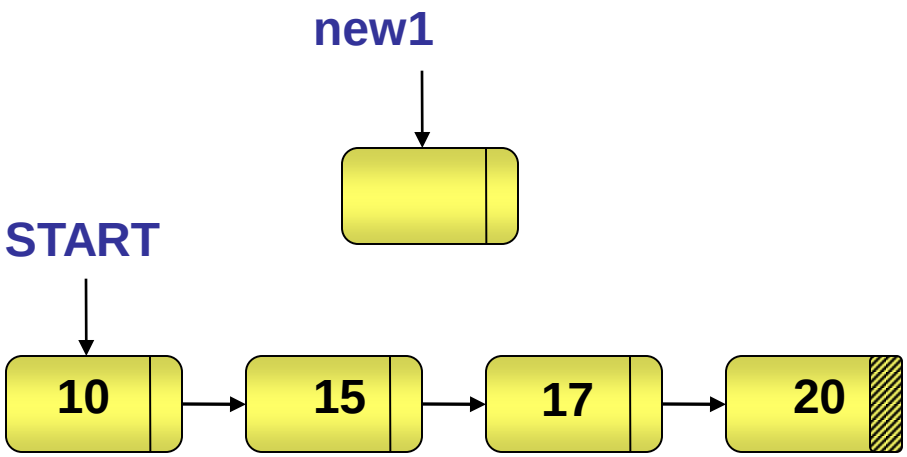
```
Algorithm InsertAtSpec()
{
1. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
2. Enter Data and Location
3. new1->info=Data
4. If (Location == 1)
        4.1 new1 -> next = Start
        4.2 Start = new1
    Else
        4.1 Previous = Start
        4.2 Count = 1
        4.3 While(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

Write an algorithm to insert a node in a Sorted linked list.

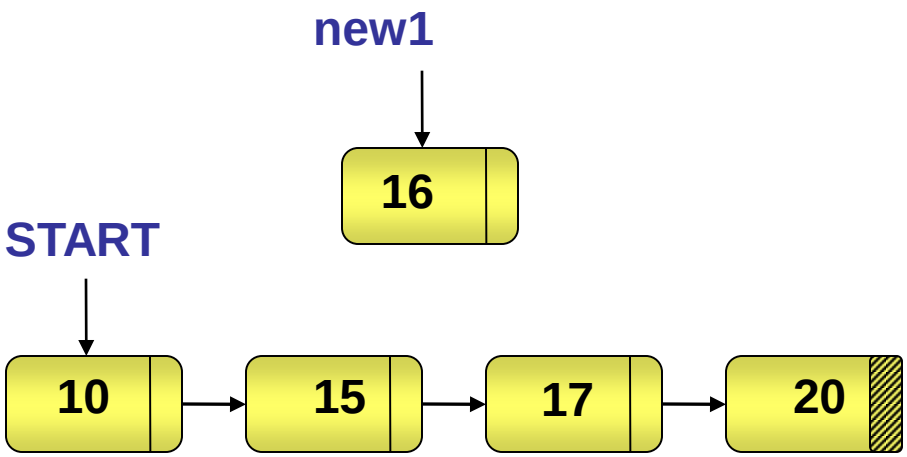
Insert 16



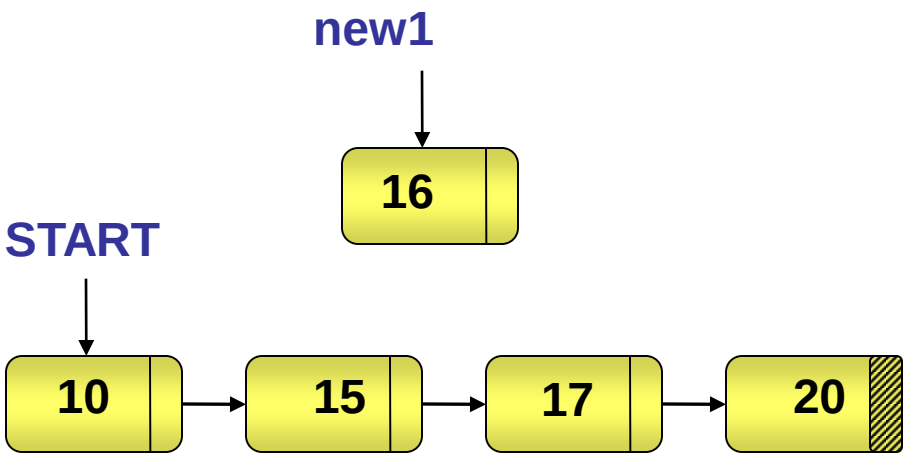
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



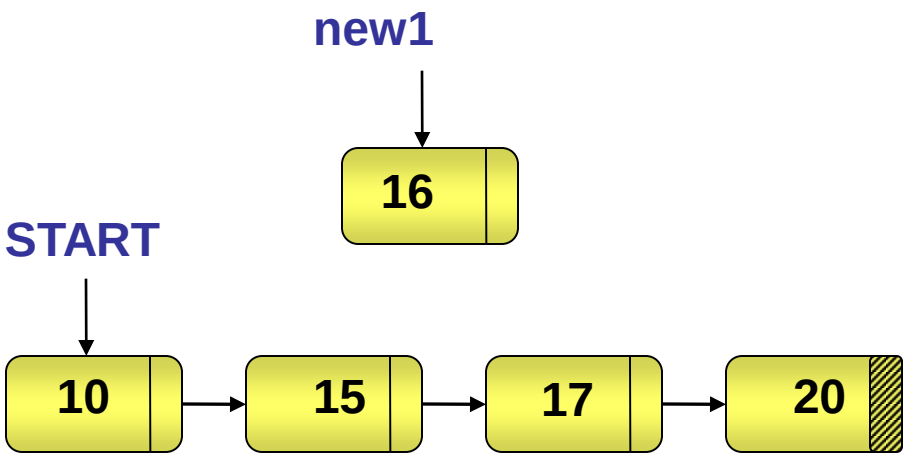
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



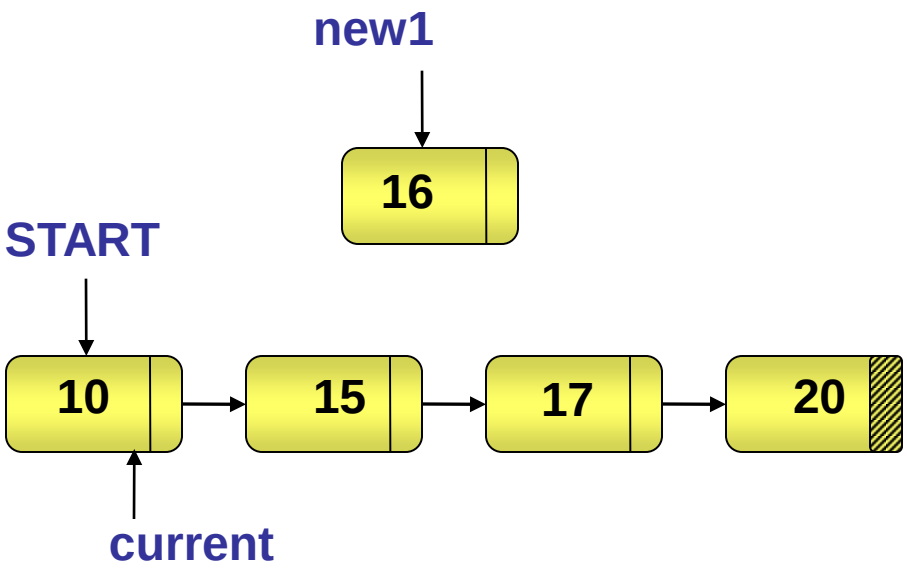
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



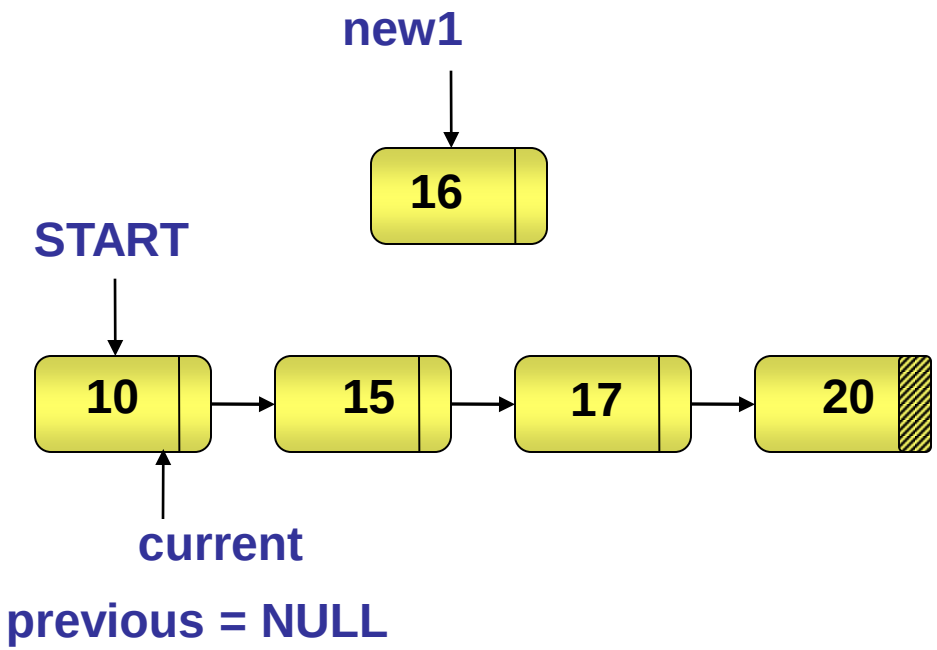
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



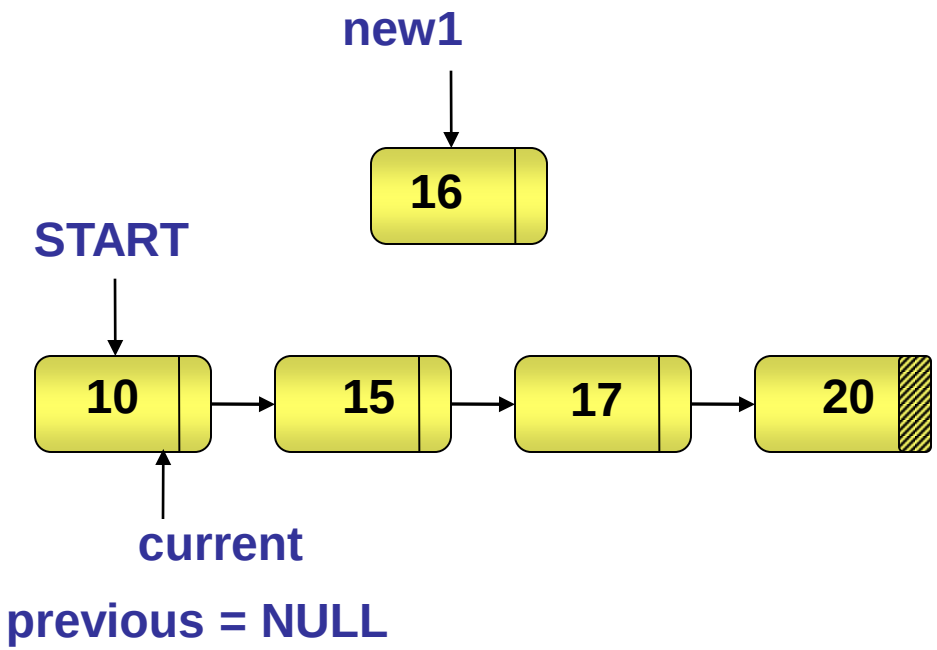
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



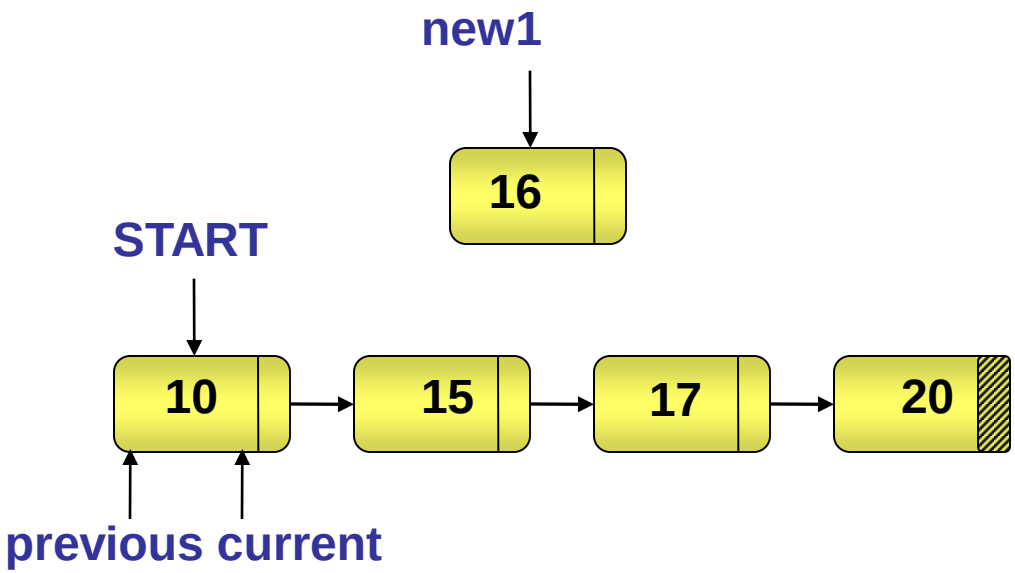
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



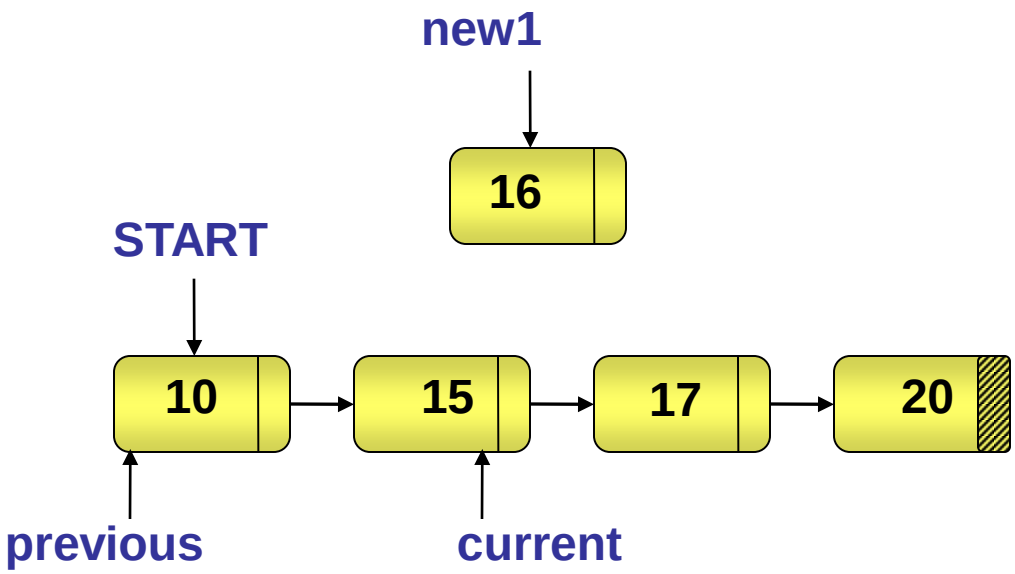
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



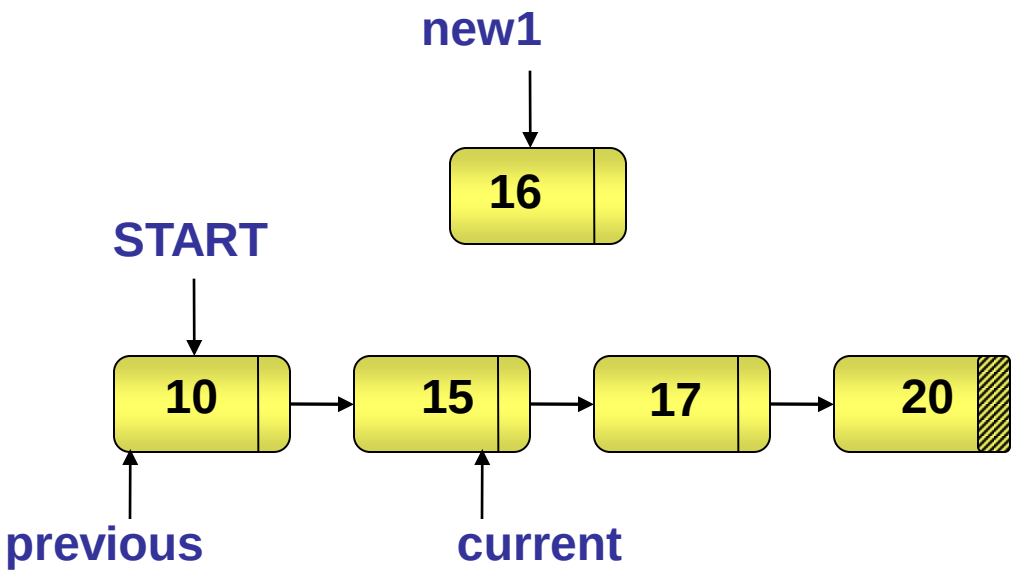
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



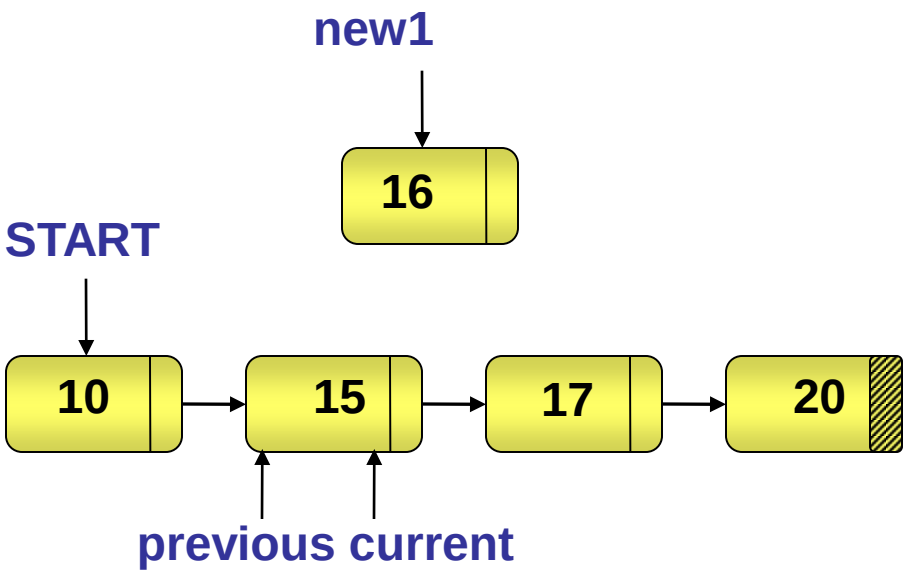
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



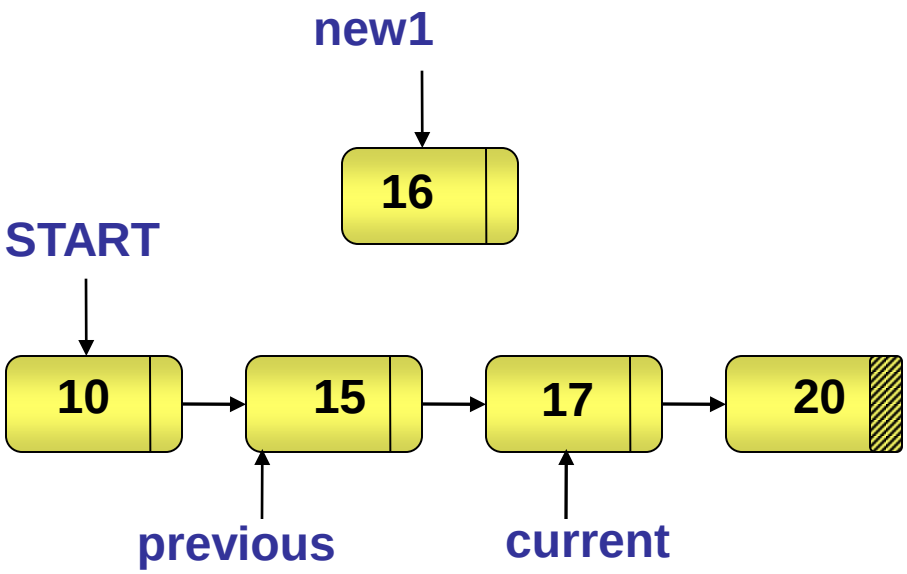
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



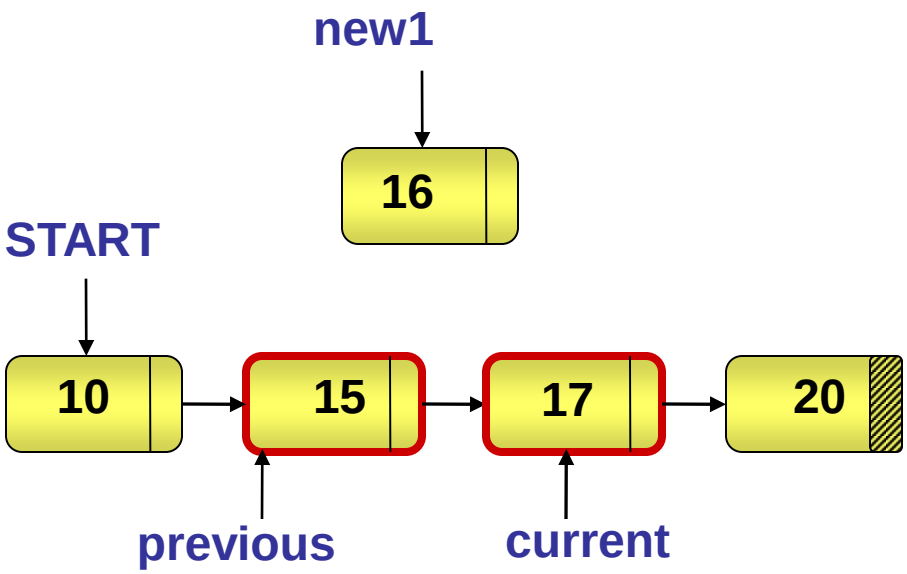
1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.

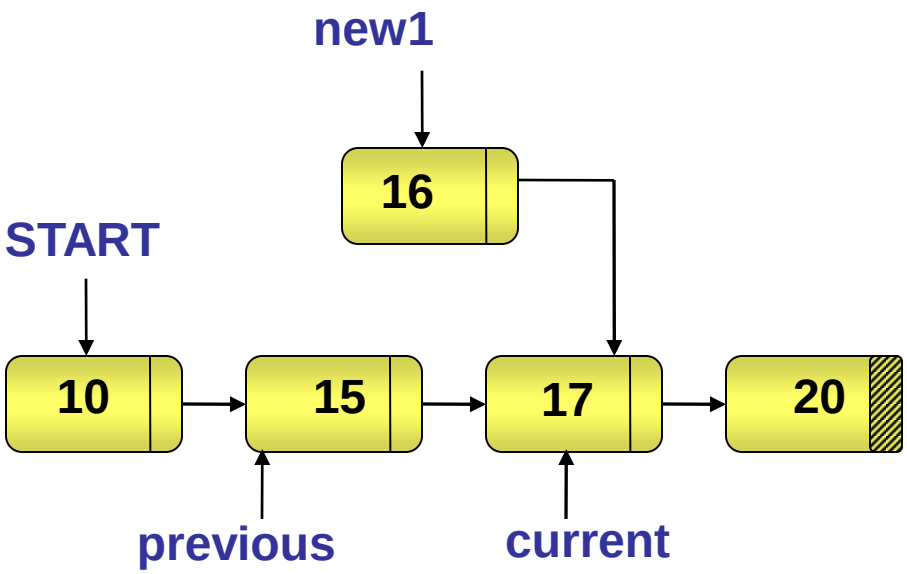


1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



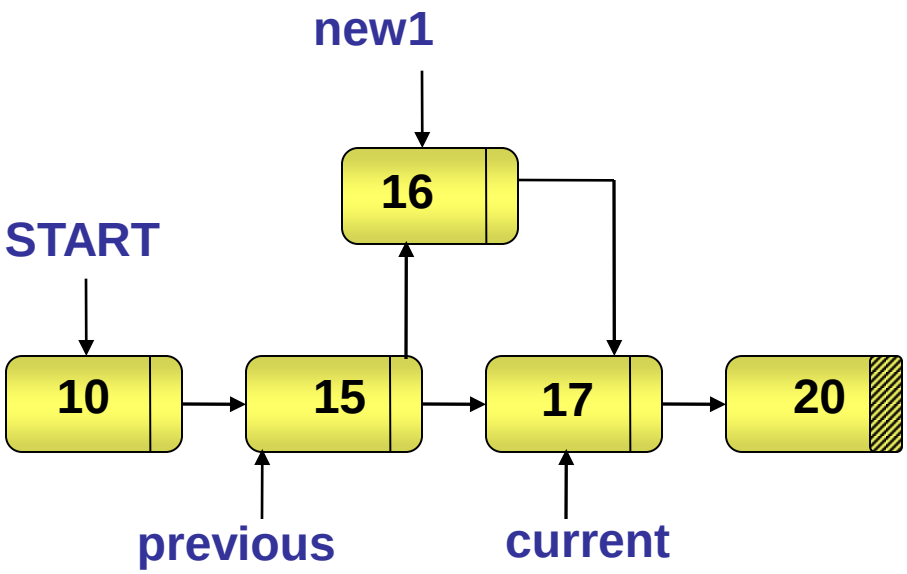
Nodes located

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



new1 -> next = current

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.

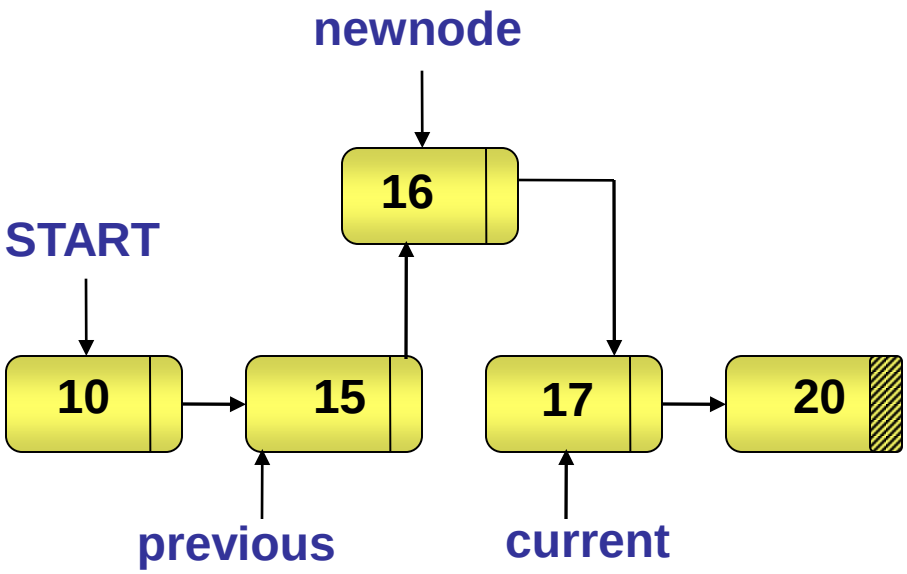


new1 -> next = current

previous -> next = new1

Insertion complete

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.



new1 -> next = current

previous -> next = new1

Insertion complete

1. Allocate memory for the new node.
2. Assign value to the data field of the new node.
3. If data of newnode is less than the data of start node, Then
   a. make newnode point to the start node
   b. make start point to the newnode
4. Otherwise,Identify the nodes between which the new node is to be inserted. Mark them as previous and current. To locate previous and current, execute the following steps:
   a. Make current point to the first node.
   b. Make previous point to NULL.
   c. Repeat step d and step e until current.info becomes greater than newnode.info or current becomes equal to NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.
5. Make the next field of the new node point to current.
6. Make the next field of previous point to the new node.

ALGORITHM TO INSERT NODE IN A SORTED LINKED LIST

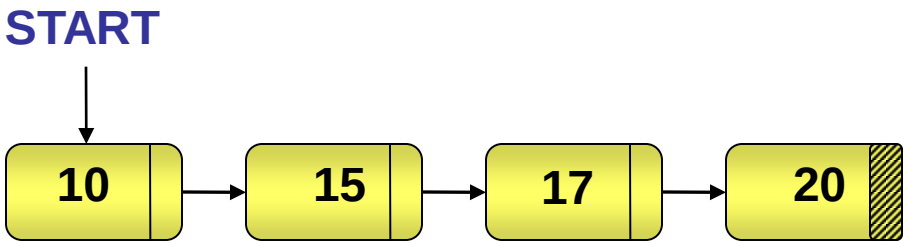
```
Algorithm InsertAtSorted()
{
1. Enter Data
2. Create node [(new1=(struct node*) malloc(sizeof(struct node))]
3. new1->info=Data
4. If (Data <= Start -> info)
        4.1 new1 -> next = Start
        4.2 Start = new1
    Else
        4.1 Current = Start
        4.2 Previous = NULL
        4.3 While(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

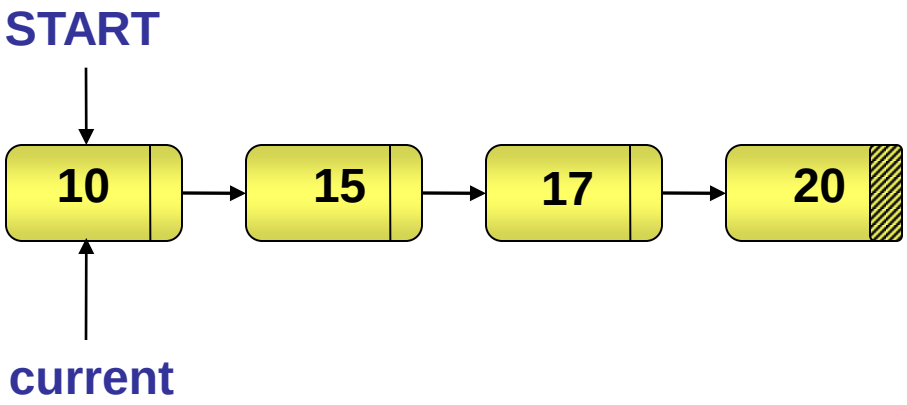
Deleting a Node from a Singly-Linked List

- Delete operation in a linked list refers to the process of removing a specified node from the list.
- You can delete a node from the following places in a linked list:
  - Beginning of the list
  - Between two nodes in the list
  - End of the list

- Algorithm to delete a node from the beginning of a linked list.

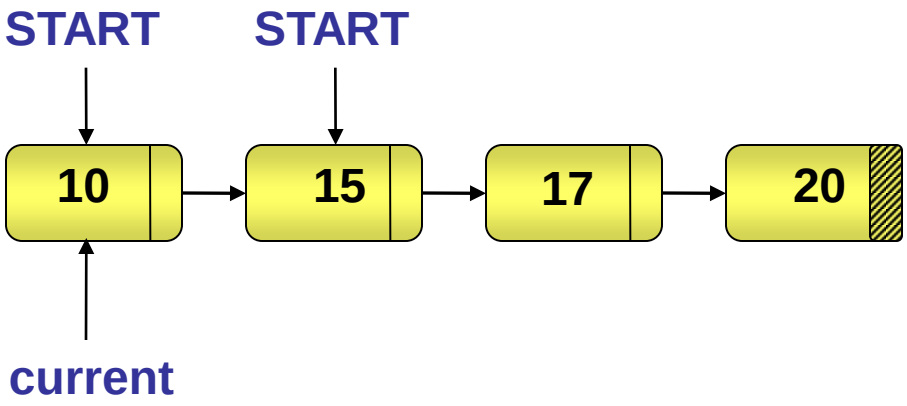


1. Mark the first node in the list as current.
2. Make START point to the next node in its sequence.
3. Release the memory for the node marked as current.



1. Mark the first node in the list as current.
2. Make START point to the next node in its sequence.
3. Release the memory for the node marked as current.

current = START



1. Mark the first node in the list as current.
1. Make START point to the next node in its sequence.
1. Release the memory for the node marked as current.

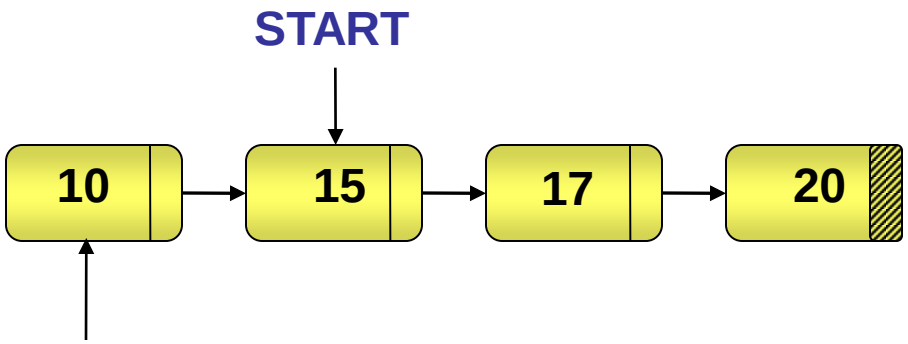
current = START

START = START -> next

1. Mark the first node in the list as current.

1. Make START point to the next node in its sequence.

1. Release the memory for the node marked as current.



Memory released

current = START

START = START -> next

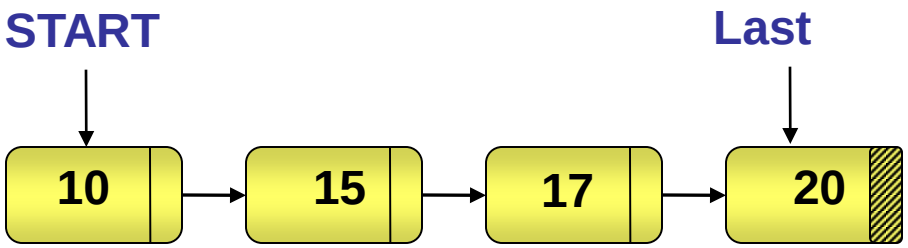
Delete operation complete

ALGORITHM TO DELETE A NODE FROM THE BEGINING

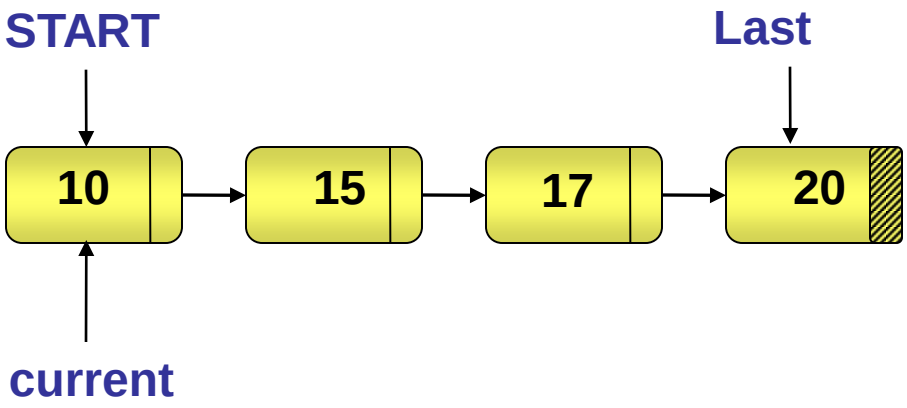
```
Algorithm DeleteAtBeg()
{
1. If (Start == NULL)
            1.1 Print "underflow“
   else
            1.1 Current = Start
            1.2 Start = Start -> next
            1.3 Current -> next = NULL
            1.4 Release the memory [ free (Current) ]

}
```

- Algorithm to delete a node from the End.

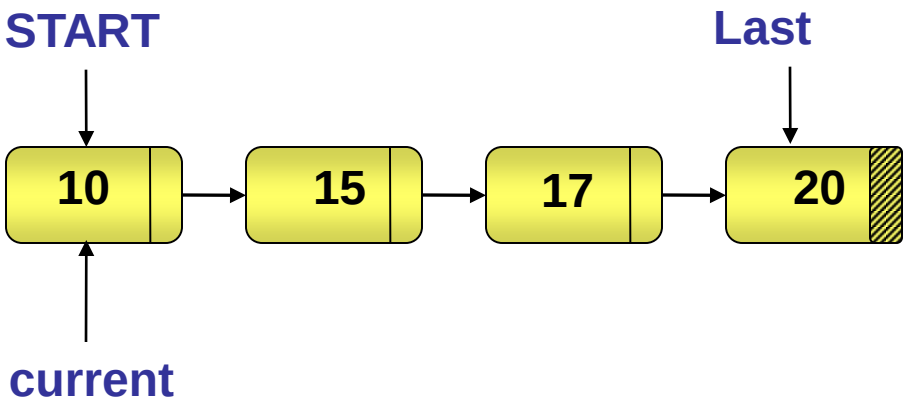


1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node

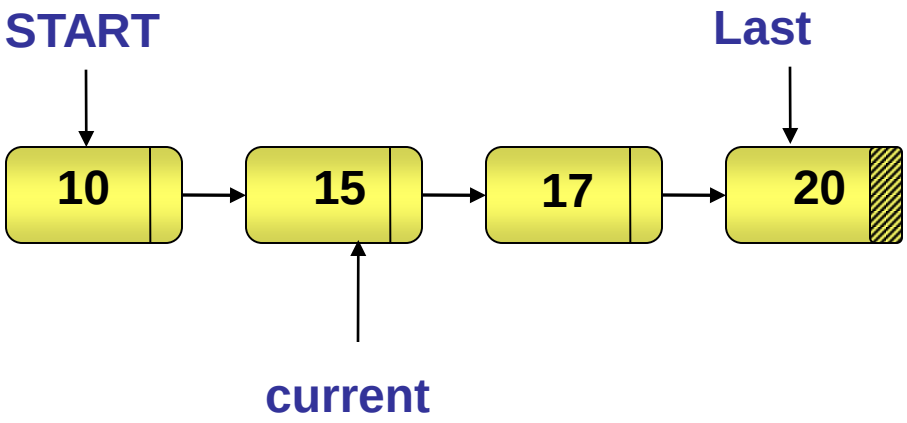


current = START

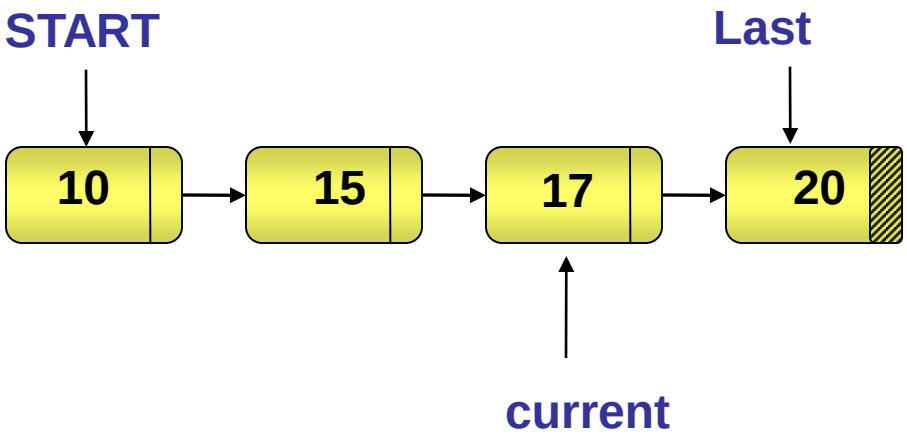
1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node

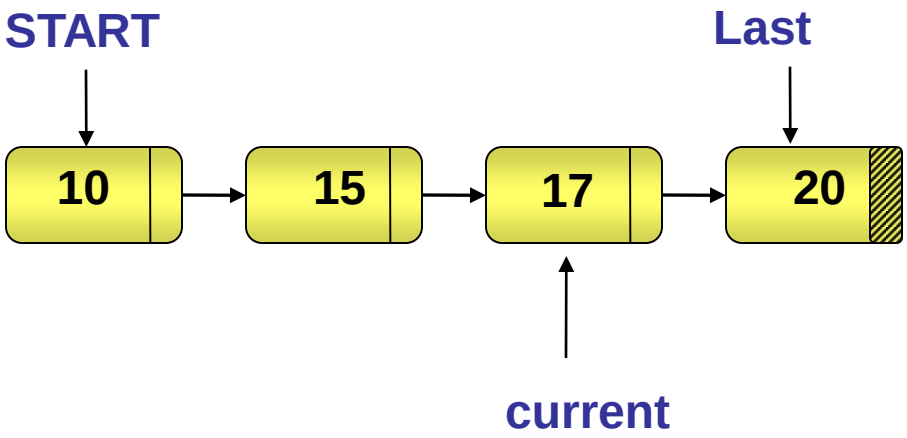


1. Mark the first node in the list as current..
2. Repeat step 3 untill current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node

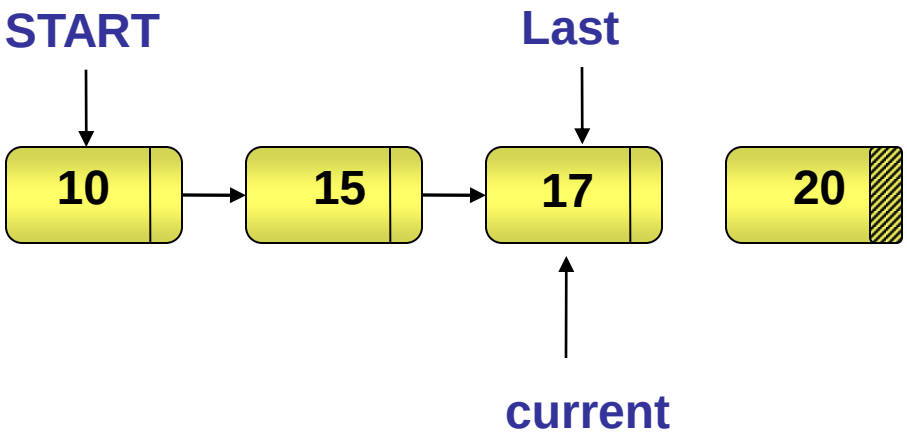


1. Mark the first node in the list as current..
2. Repeat step 3 untill next of current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node


START
Last
10
15
17
20
current



1. Mark the first node in the list as current..
2. Repeat step 3 untill next of current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node



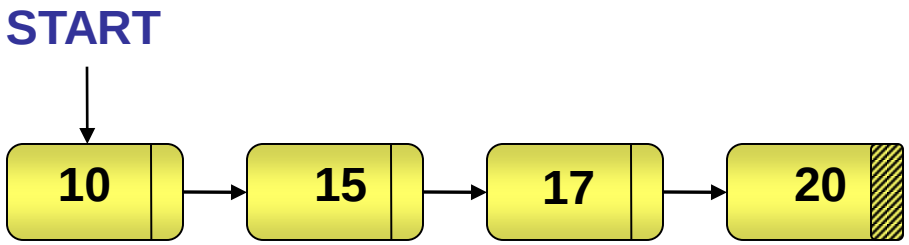
1. Mark the first node in the list as current..
2. Repeat step 3 untill next of current becomes Last node.
3. Make current point to the next node in its sequence.
4. Release the memory for the node marked as Last.
5. Make Current point to the last node

Delete operation complete

ALGORITHM TO DELETE A NODE FROM THE END

```
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 to delete a node from a specific position.
- Delete 17

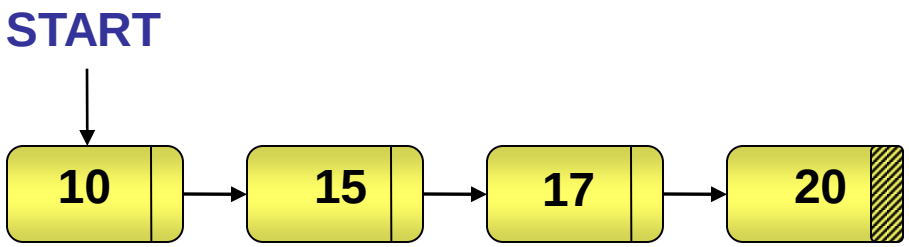


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current .
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

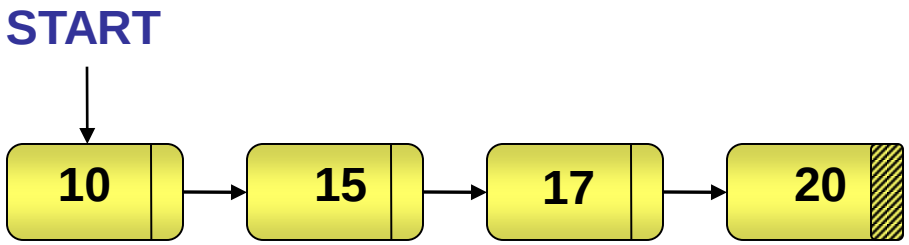


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17



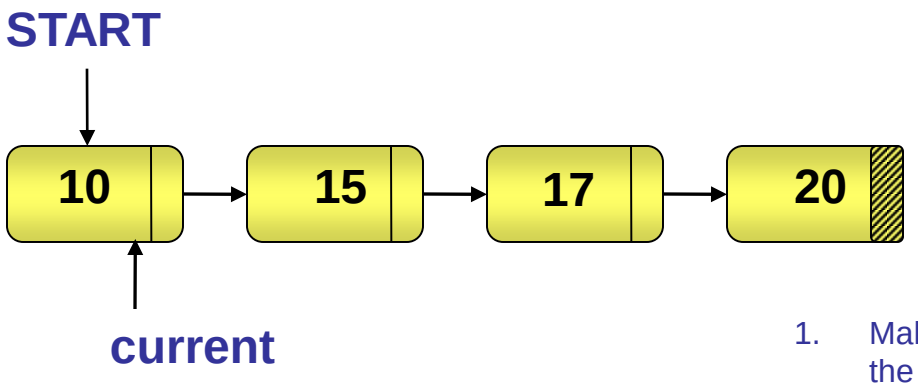
Previous = NULL

1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17



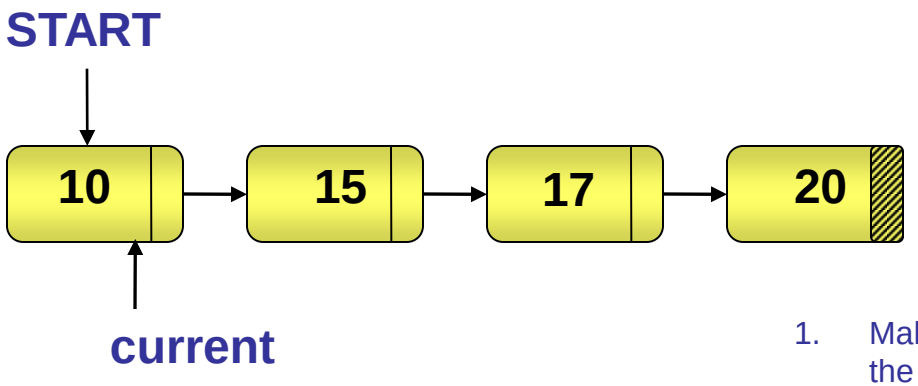
Previous = NULL

1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17



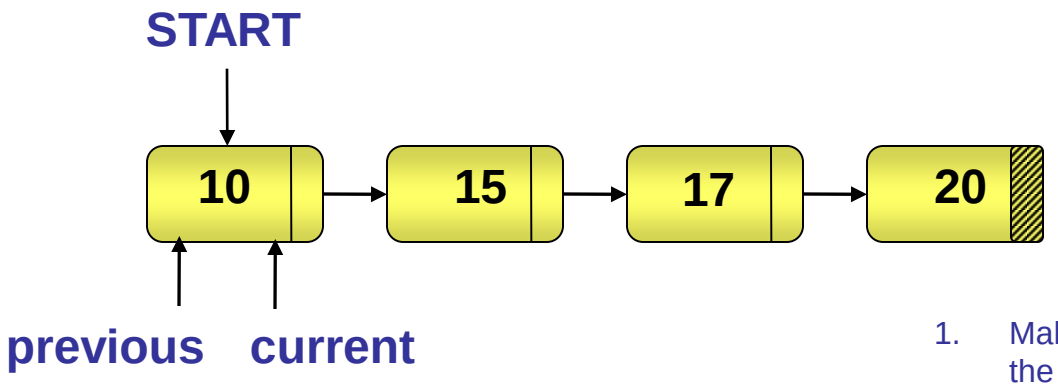
Previous = NULL

1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

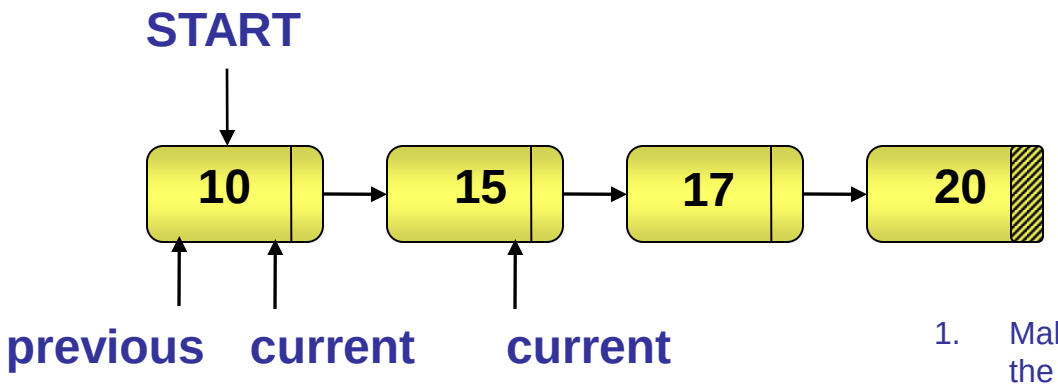


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

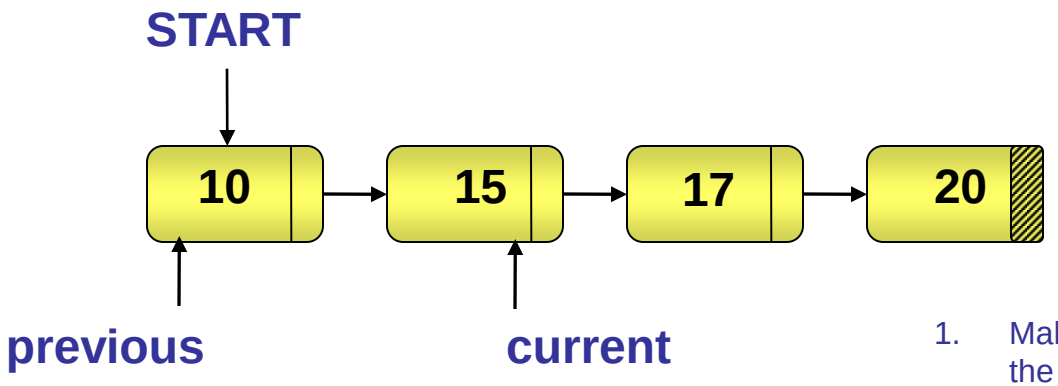


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
   a. Set previous = NULL
   b. Set current = START
   c. Repeat step d and e until either the node is found or current becomes NULL.
   d. Make previous point to current.
   e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

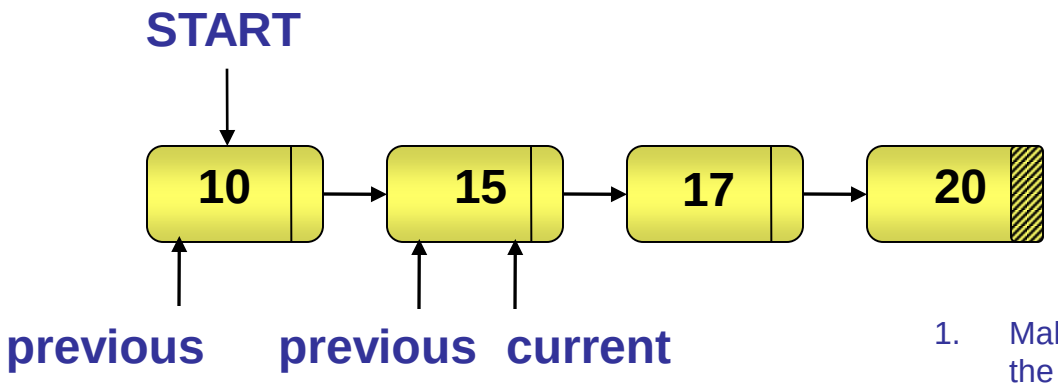


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

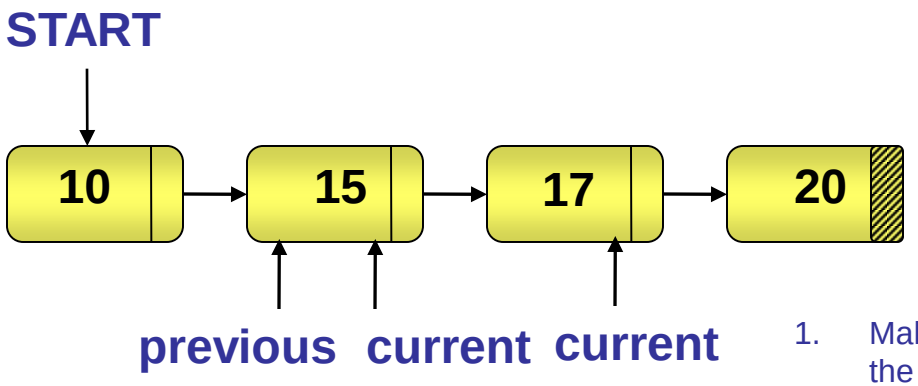


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

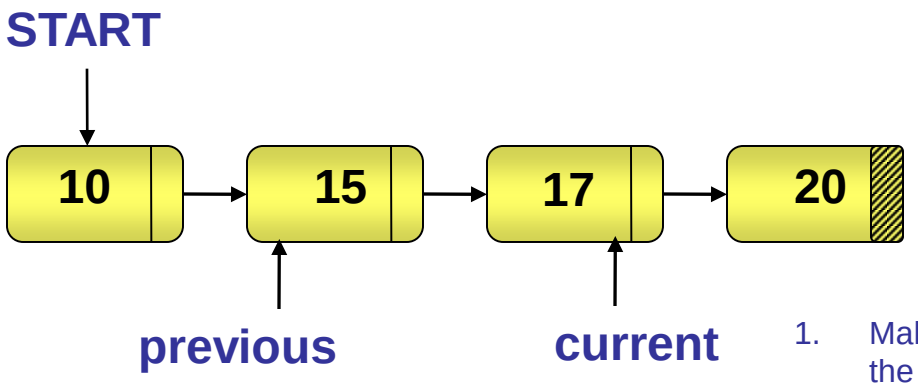


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17

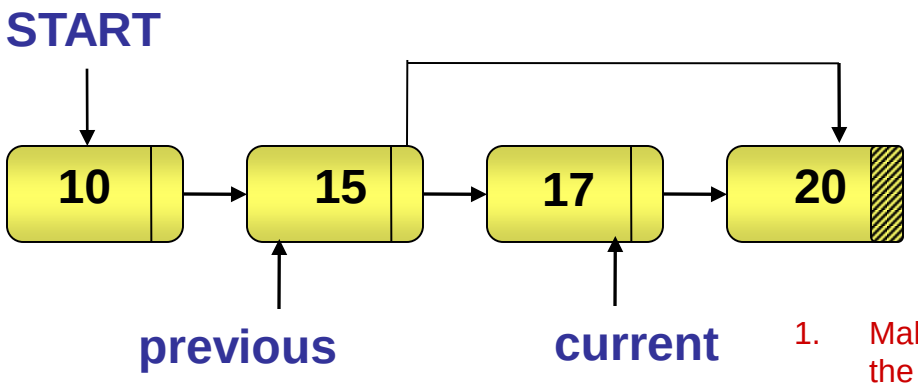


1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

- Delete 17



1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

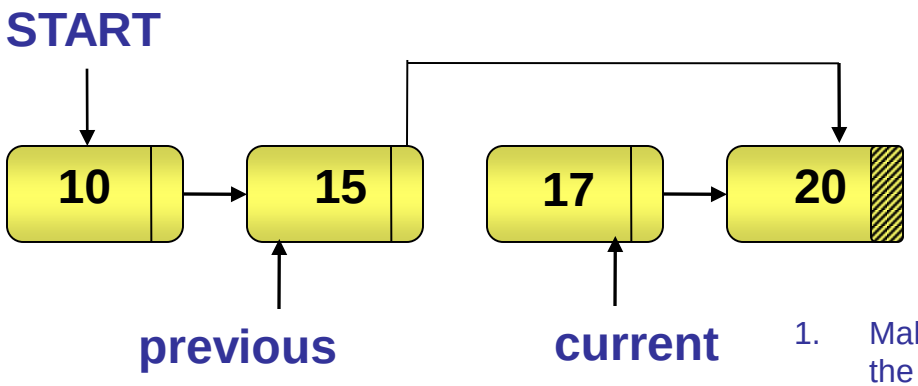
1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

Previous -> next = current -> next

- Delete 17

Delete operation complete



1. Locate the node to be deleted. Mark the node to be deleted as current and its predecessor as previous. To locate current and previous, execute the following steps:
    a. Set previous = NULL
    b. Set current = START
    c. Repeat step d and e until either the node is found or current becomes NULL.
    d. Make previous point to current.
    e. Make current point to the next node in sequence.

1. Make the next field of previous point to the successor of current.

1. Release the memory for the node marked as current.

Previous -> next = current -> next

ALGORITHM TO DELETE A NODE FROM THE SPECIFIC POSITION

```
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) ]
}
```

- Problem Statement
  - Discuss the advantages and disadvantages of linked lists.

- Problem Statement
  - Discuss the differences between arrays and linked lists.

- Linked lists allow _____ access to elements.

- Answer:
  - sequential

Printing Reverse the list

printReverse(head)

1. call printReverse(head->next)
2. print head->data

```
void printReverse(Node* head)
{
    // Base case
    if (head == NULL)
    return;
    // print the list after head node
    printReverse(head->next);
      // After everything else is printed, print head
    cout << head->data << " ";
}
```

Actual reverse of list

```
void reverse()
  {
      // Initialize current, previous and
      // next pointers
      Node* current = head;
      Node *prev = NULL, *next = NULL;
       while (current != NULL) {
         // Store next
         next = current->link;
           // Reverse current node's pointer
         current->link = prev;
           // Move pointers one position ahead.
         prev = current;
         current = next;
      }
 head = prev;
```

Sort the List

```
 void sort(){ if(head == NULL) {   return;  }
else {
        while(current != NULL) {
            //Node index will point to node next to current
            index = current->next;
              while(index != NULL) {   //If current node's data is greater than index's node data, swap the data between them
                if(current->data > index->data) {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;  }
                index = index->next;
            }  current = current->next; }     }
```