CHAPTER 5

Linked Lists

Introduction

As we show in chapter 4 that data in an array are stored by continuous memory location.

There were several problems of arrays:

1. to insert or to delete data other than at the end.
2. to grow or to shrink the length of a list.
3. It use consecutive block of memory space. We may have enough memory but if they are not consecutive, program can not run.
4. It is static data structure. i.e. list size must be known in advance.

Advantages & Disadvantages of Linked List

Those problems of arrays can be overcome by another type of data structure called Linked List.

A Linked List, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of Pointers.

Here each node has two parts: the first part, called the information part, contains the information of the elements and the second part, called the link field or next pointer field, contains the address of the next node in the list.

Advantages & Disadvantages of Linked List

How ever Linked List have some disadvantages too.

1. It is complicated for beginners to understand.

2. In simple and small structure linked list requires extra memory spaces for the pointer fields, but is negligible for large and complicated structure.

3. It is not possible to access any arbitrary elements directly. (For this reason Binary Search Algorithm can not be employed here)

Linked List Diagram

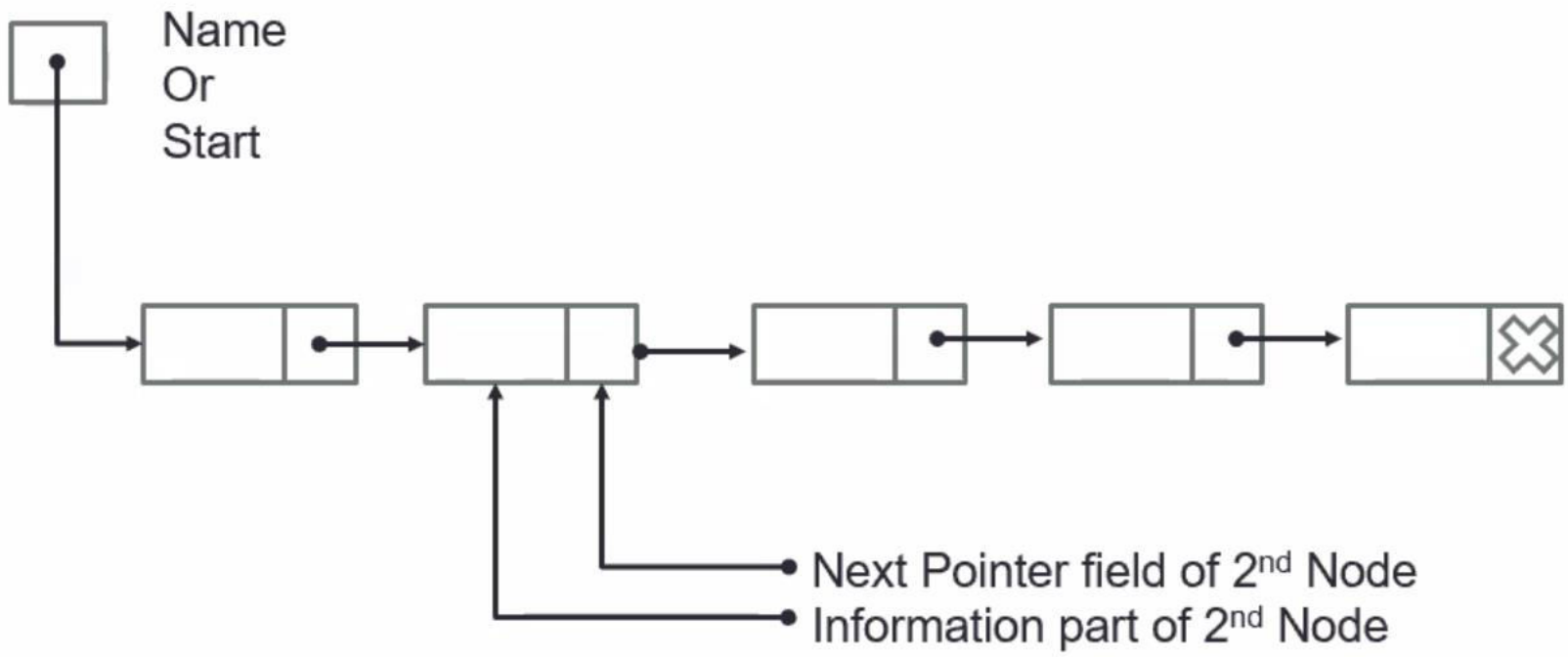
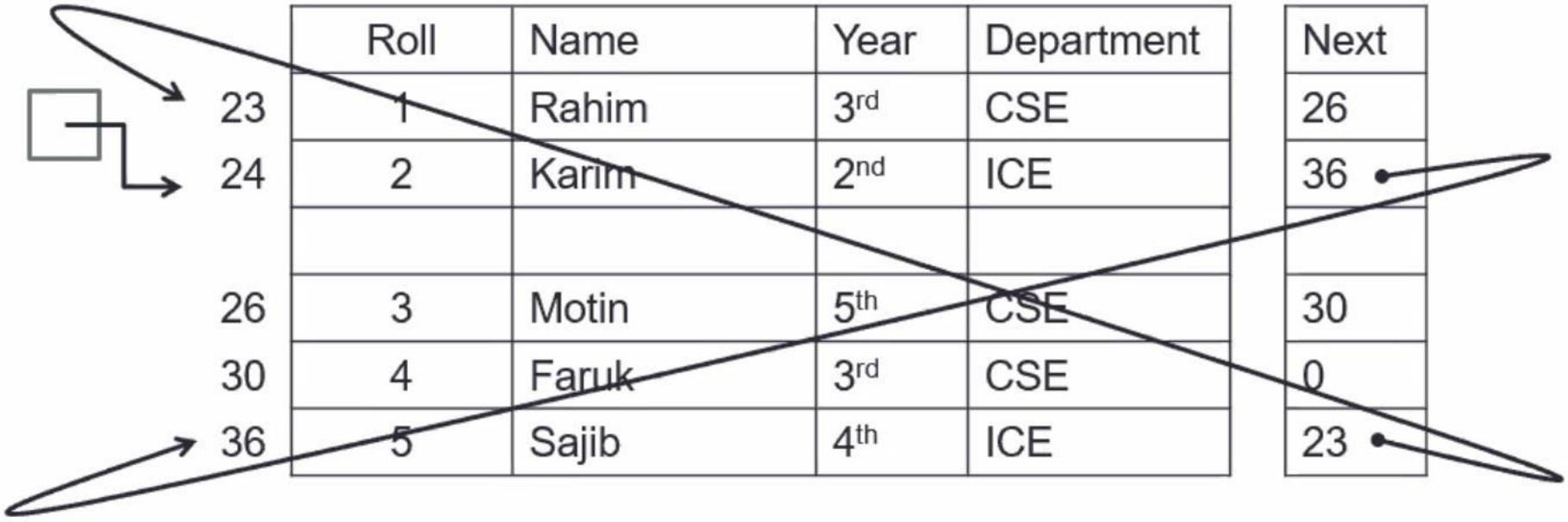


Fig: A Typical Linked List

Linked List Example

Information field of a Linked List can be a Integer Valued Data, String, or Records. In the following example we show a Record of Student's Roll, Name, Year, Department as information part.



	Roll	Name	Year	Department	Next
23	1	Rahim	3rd	CSE	26
24	2	Karim	2nd	ICE	36
26	3	Motin	5th	CSE	30
30	4	Faruk	3rd	CSE	0
36	5	Sajib	4th	ICE	23

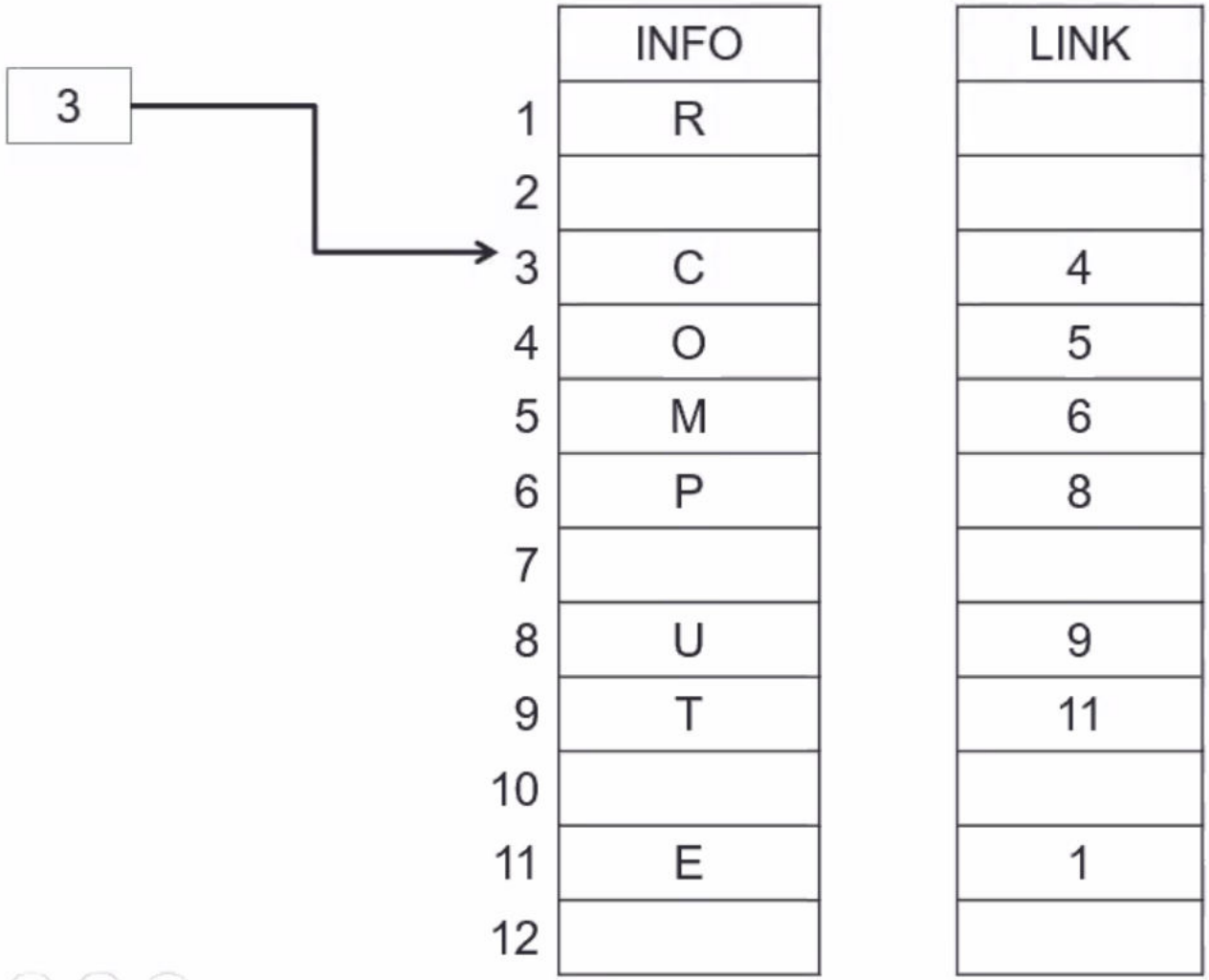
C Code to declare a node

```
struct node{
        int roll;
        char name[20];
        char year[3];
        char department[40];
        struct node *next;   //Next pointer field
        };
typedef struct node *nodeptr;  //user defined data type
// You can use nodeptr as a new type declare variable of this type.
```

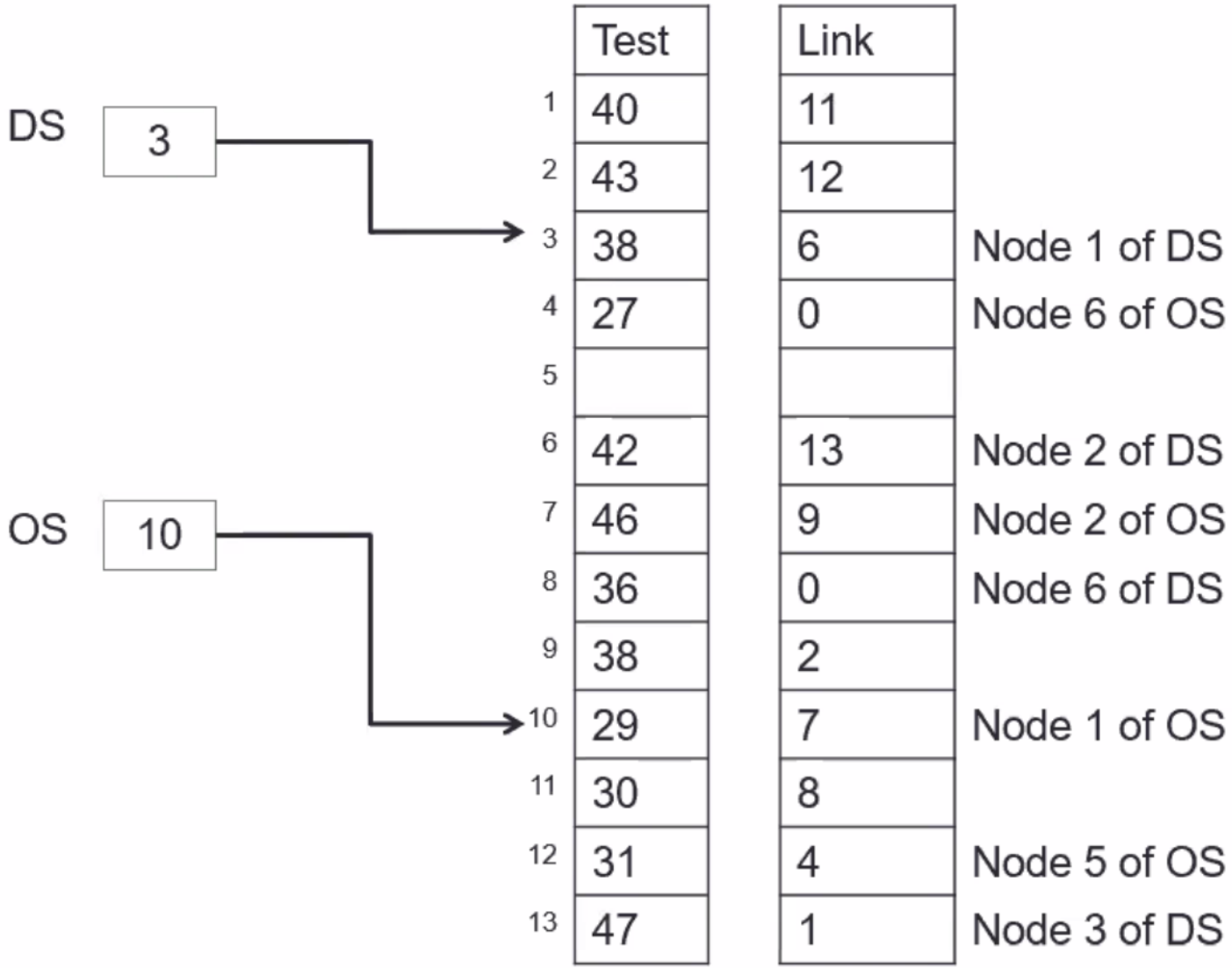
Representation of Linked List in Memory

- The List contains two linear arrays: info and link.
- Info[k] contains the information of node k.
- Link[k] contains the nextpointer field of node k.
- As we show in the next example: nodes need not occupy adjacent elements in the array info and link.
- More than one list may be maintained in the same linear arrays info and link.
- However, each list must have its own pointer variable giving the location of its first node.

Memory Representation Example

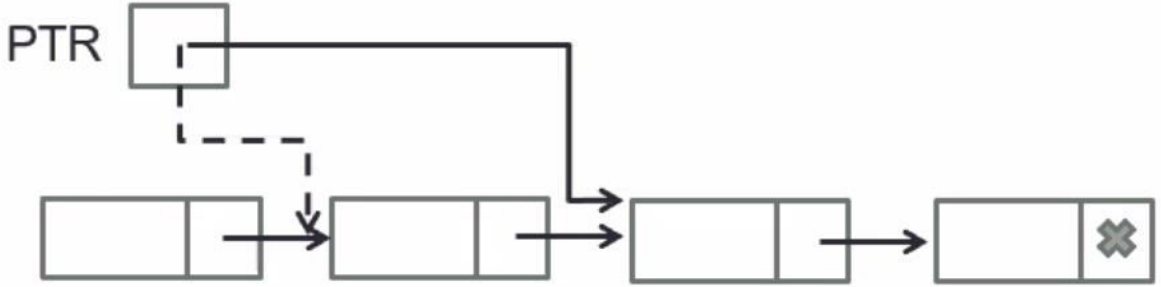


Two List in the same Array



Traversing a Linked List

- Let LIST be a linked list stored in linear arrays info and link with START pointing to the first element and NULL indicating the end of LIST.
- We use PTR as a pointer variable which points to the node that is currently being processed.
- Thus link[PTR] points to the next node to be processed.
- PTR:=link[PTR] moves the pointer to the next node in the list as the picture shows below.



Traversing a Linked List

- This algorithm Traverse LIST by applying an operation Process to each of element of LIST.

1. Set ptr:=start //[Initialize Pointer PTR]
2. Repeat steps 3 and 4 while ptr≠NULL
3. Apply PROCESS to info[ptr]
4. Set ptr:=link[ptr] //[ptr now points to the next node]
5. Exit

C Code to Create a Node to Memory

In C there is a function malloc which can be used to dynamically assign memory to a variable. It returns a pointer to the reserved block of memory (if there is any free memory in the system).

```
nodeptr getnode(void) //This function returns a Node
{
        nodeptr p; //Node Type variable p
        p=(nodeptr) malloc(sizeof(struct node));
                //Reserve a block of memory to node p
        return(p); //Return Node Pointer
}
```

C Code to Traverse(Average) the List

```
ptr=start->next;
sum=0;
count=0;
while(ptr!=0)
{
        sum=sum+ptr->info;
        count++;
        //printf("%d ",ptr->info);
        ptr=ptr->next;
}
avg=sum/count;
```

Searching a Linked List

- Searching a List means finding out the location of a particular item in that list.
- The list can be in Unsorted or Sorted.

- However we can not employ Binary Search on Linked List as is done on array because there is no way of indexing the middle element of the list directly.

- For unsorted list, we have to check two things:

  (1) ptr=NULL (2) info[ptr]=item

- For sorted list, we have to check three things:

  (1) ptr=NILL (2) item<info[ptr] and (3) item=info[ptr]

Searching in Unsorted List

SEARCH (info, link, start, item, loc)

This algorithm finds the location LOC of the node where item first appears in list, or set loc=NULL.

1. Set ptr:=Start
2. Repeat steps 3 while ptr≠NULL
3. If item=info[ptr] then:
   Set loc:=ptr and Exit. [Search is Successful]
   Else:
   Set ptr:=link[ptr] [ptr now points to next node]
4. Set loc:=NULL [Search is Unsuccessful]
5. Exit

Searching in Sorted List

SRCHSL(info, link, start, item, loc)

This algorithm finds the location loc of the node where item first appears in the sorted LIST, or sets loc=NULL.

1. Set ptr:=start
2. Repeat step 3 while ptr ≠NULL
3. If item>info[ptr] then:
   Set ptr:=link[ptr]
   Else if item=info[ptr] then:
   Set loc:=ptr and Exit [Search is Successful]
   Else:
   Set loc:=NULL and Exit [Item now exceeds info[ptr].]
4. Set loc:=NULL
5. Exit

Garbage Collection

- When a node is deleted from a list, the deleted node should be put onto a list of available space or free storage list or free pool so that another can use that.
- The OS often periodically collect all the deleted space onto free-storage list.
- The technique to collect the unused space is called Garbage Collection.

Garbage Collection

- Garbage collection take place in two steps:

- First the OS runs through all list, tagging those cells which are currently in use.
- Then the OS runs through the memory, collecting all untagged space onto the free storage list.

- The Garbage collection take place:
  - when there is only some minimum amount of space or
  - no space at all left in the free-storage list or
  - when the CPU is idle and has time to do the collection.
- Actually Garbage collection is invisible to the programmer.

Overflow and Underflow

- Overflow: Data insertion to a data structure in situation when there exist no available space to the free-storage list (AVAIL=NULL) generates a error. This error is called Overflow.
- The programmer may handle overflow by printing the message OVERFLOW.
- Underflow: Data deletion from a data structure in situation when there is no elements to the data structure (START=NULL) generates a error. This error is called Underflow.
- The programmer may handle underflow by printing the message UNDERFLOW.

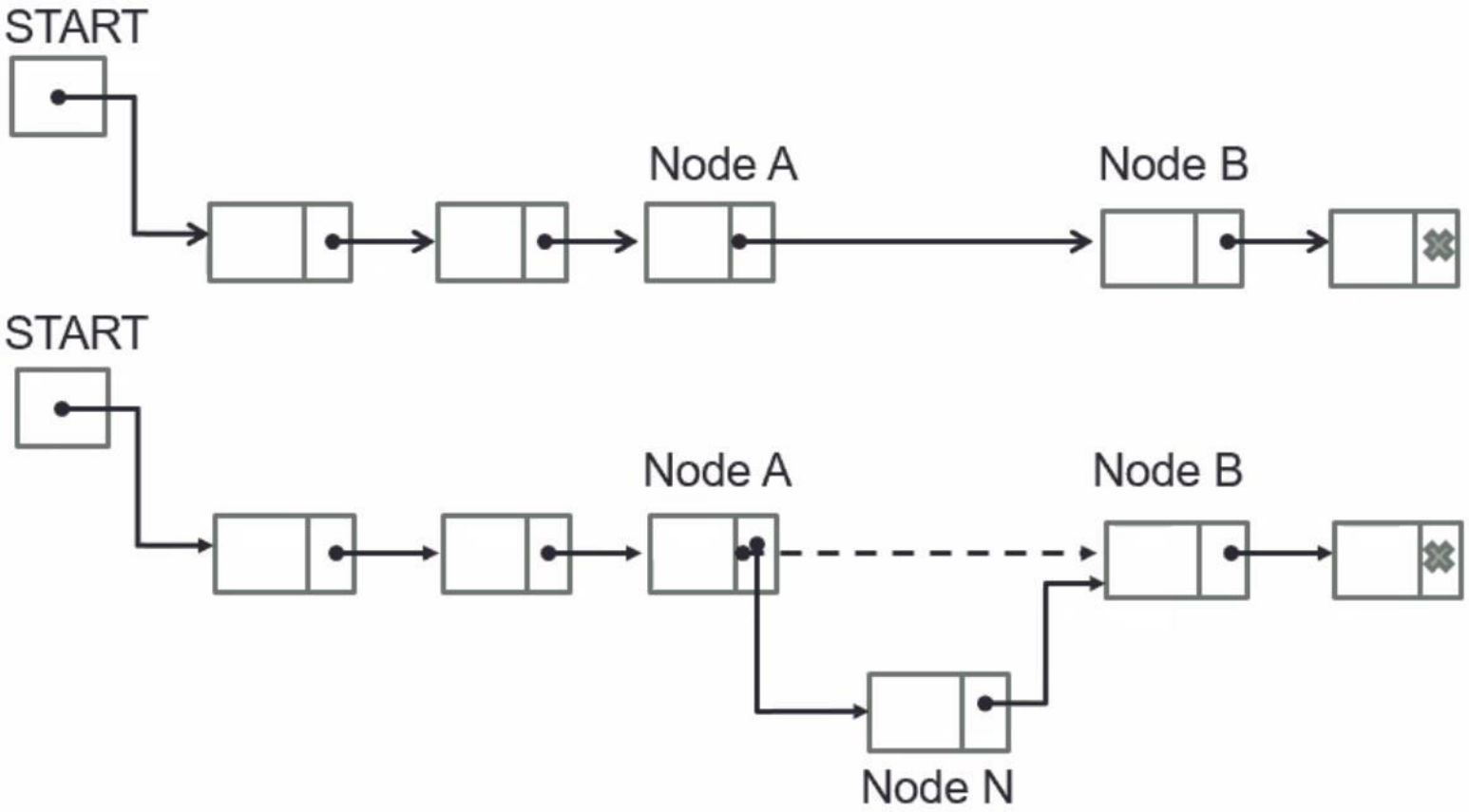
Insertion into a Linked List

- Insertion refers to add a node to a list.
- Insertion can occur in the following situations:
  1. Insertion of a node as the first node in the list.
  2. Insertion of a node following a given node in the list.
  3. Insertion of a node as the last node in the list.
  4. Insertion of a node to a Sorted list.

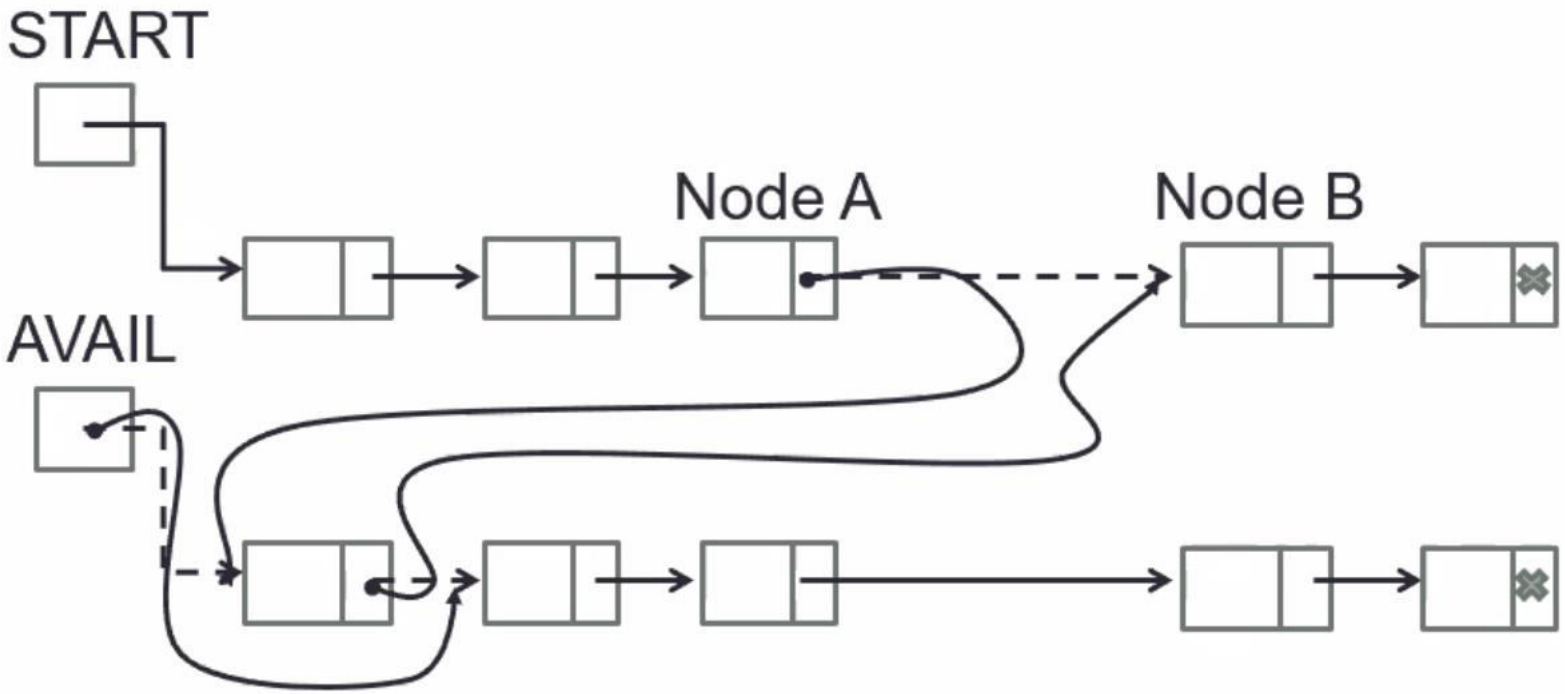
- During the insertion of a node, a new node is taken from the AVAIL list.
- How ever in C, we need only to use the malloc function to get a new node, the rest is done by the OS.

Insertion into a Linked List

- The Schematic diagram to insert node N between node A and B is shown below.



Inserting into a Linked List



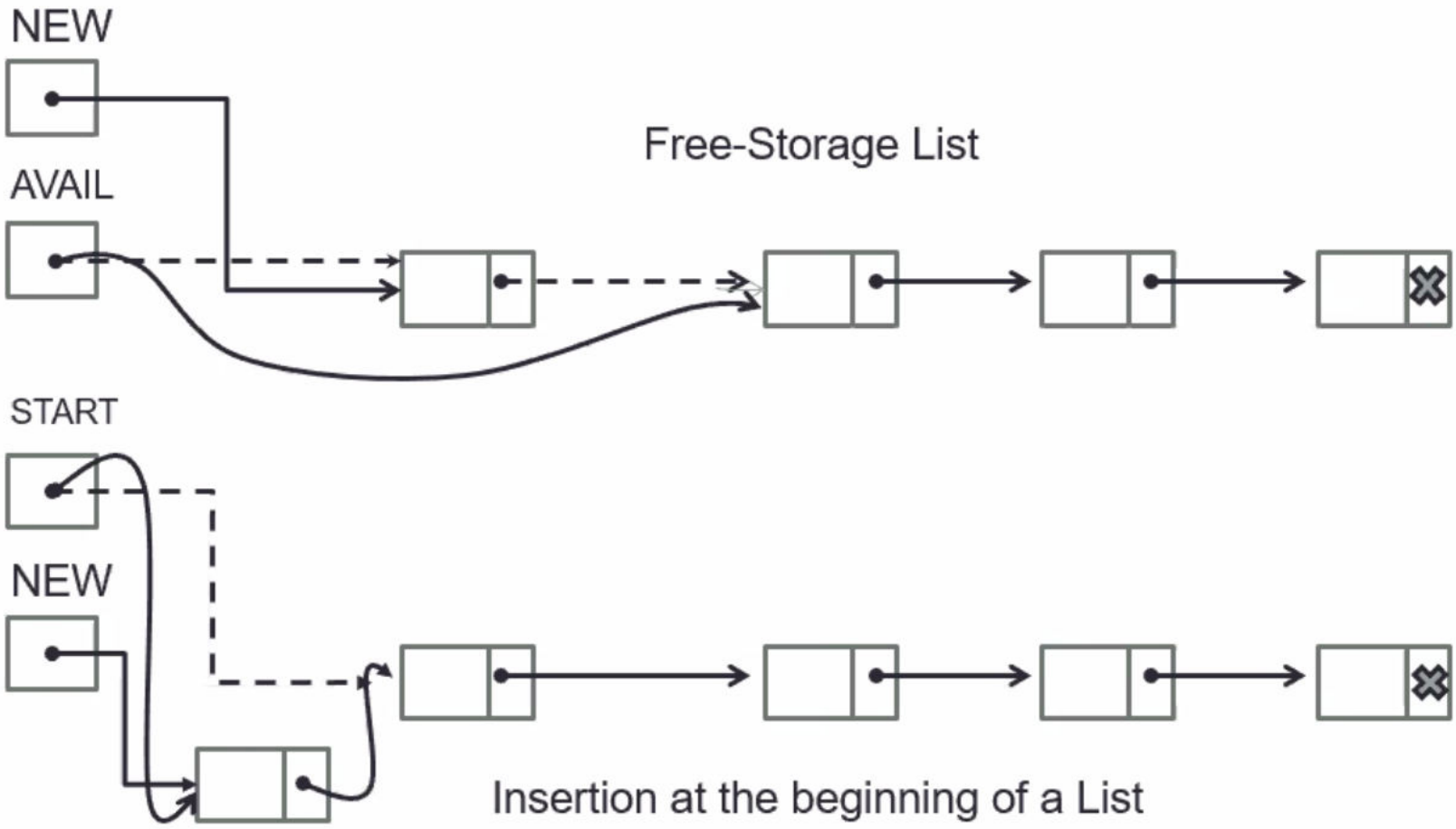
Insertion into a Linked List

- During insertion new node is taken from free-storage list.
- Three pointer fields are changed as follows:
  1. The nextpointer field of node A now points to the new node N, to which AVAIL previously pointed.
  2. AVAIL now points to the second node in the free pool, to which node N previously pointed.
  3. The nextpointer field of node N now points to node B, to which node A previously pointed.

There are also two special cases:

1. If the new node N is the first node in the list, then START will point to N.
2. If the new node N is the last node in the list, then N will contain the null pointer.

Inserting at the Beginning of a List



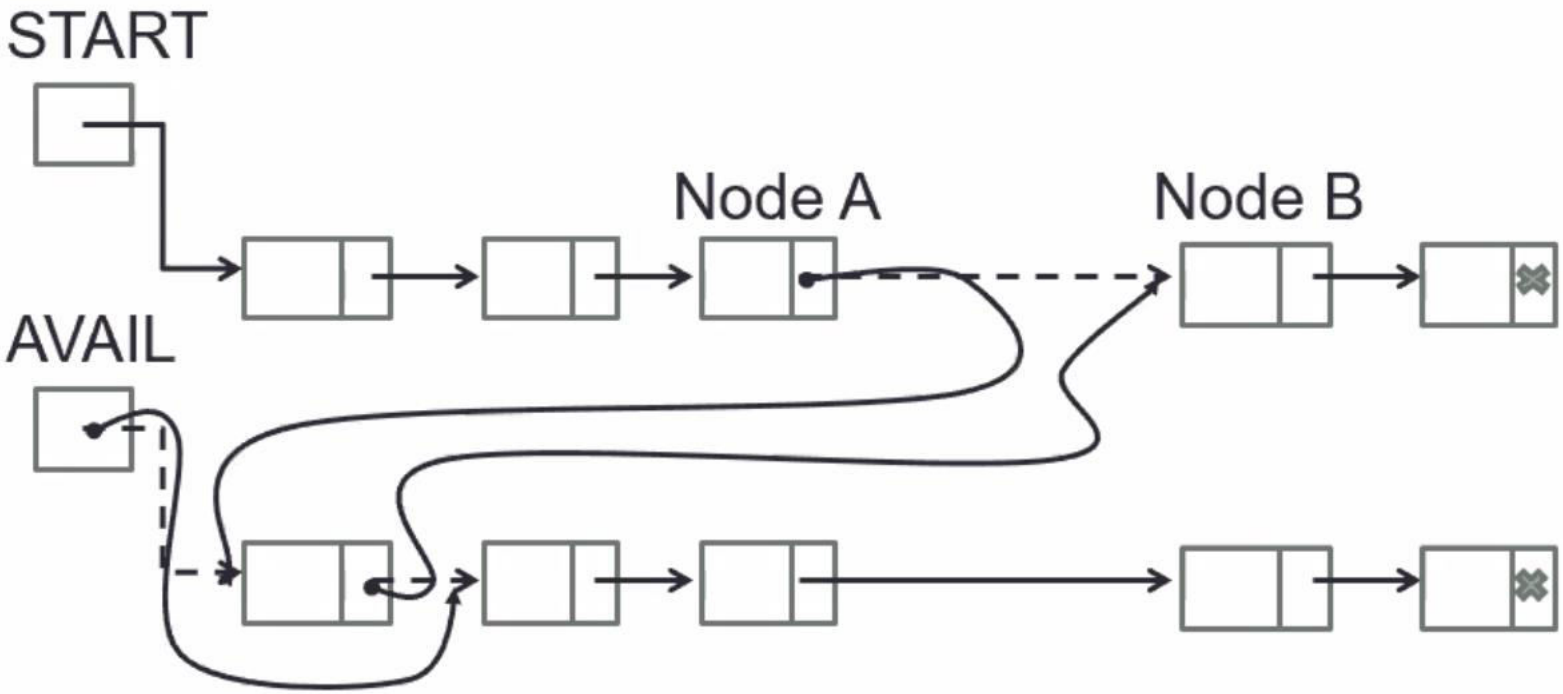
Inserting at the Beginning of a List

INSFIRST(info, link, start, avail, item)

This Algorithm inserts item as the first node in the list

1. [Overflow?] if avail=NULL, then write: OVERFLOW and Exit
2. [Remove first node from avail list]
   Set new:=avail and avail:=link[avail]
3. Set info[new]:=item [Copies new data into new node]
4. Set link[new]:=start [New node now points to original first node]
5. Set start:=new [Changes start so it points to the new node]
6. Exit

Inserting after a Given Node



Inserting after a Given Node

INSLOCK(info, link, start, avail, loc, item)

This Algorithm inserts item so that item follows the node with location loc or inserts item as the first node when loc=NULL

1. [OVERFLOW?] If avail=NULL then write: OVERFLOW and Exit.
2. [Remove first node from the avail list]
   Set New:=avail and avail:=link[avail]
3. Set info[New]:=item [Copies new data into new node]
4. If loc=NULL then: [Insert as first node]
   Set link[New]:=start and start:=new
   Else: [Insert after node with location loc]
   Set link[New]:=link[loc] and link[loc]:=New
5. Exit

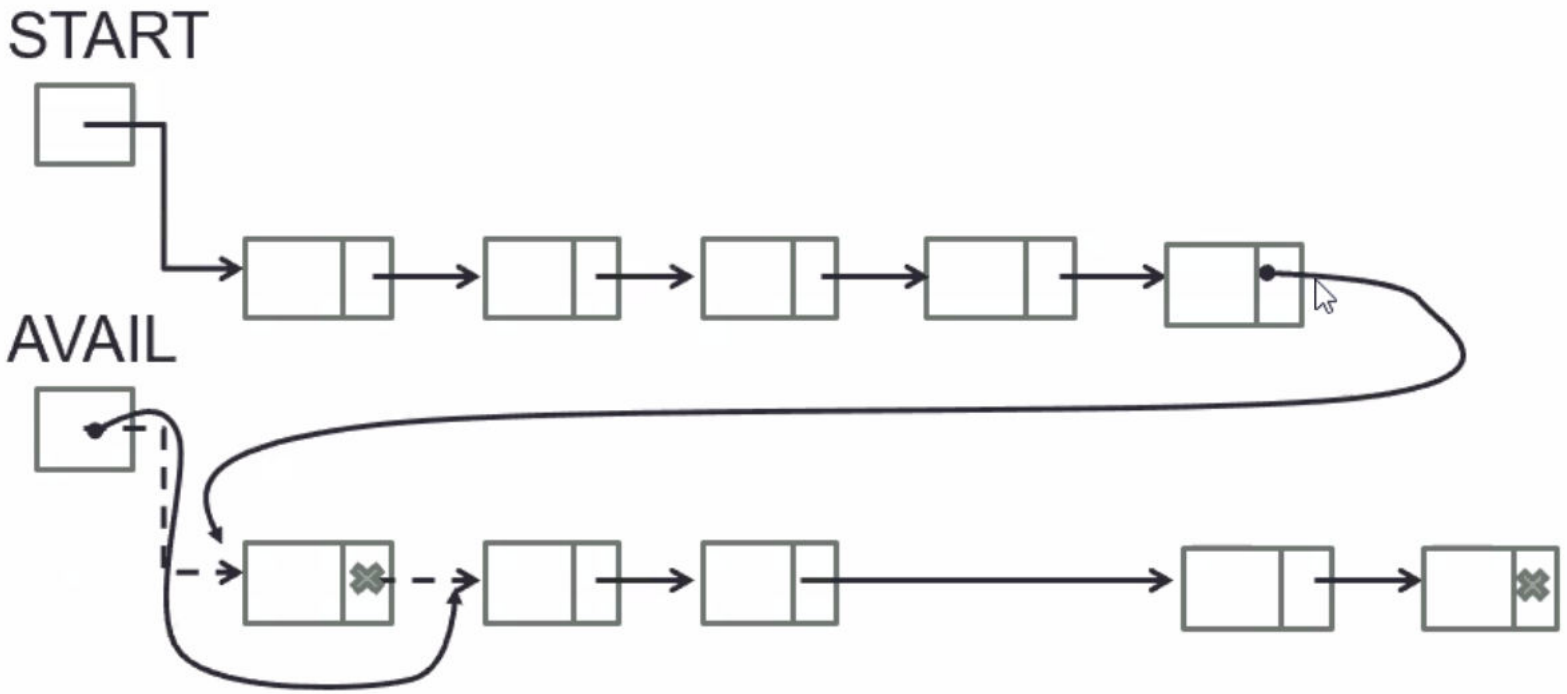
Inserting at the End

INSEND(info, link, start, avail, item)

This Algorithm inserts item as the last element of a list.

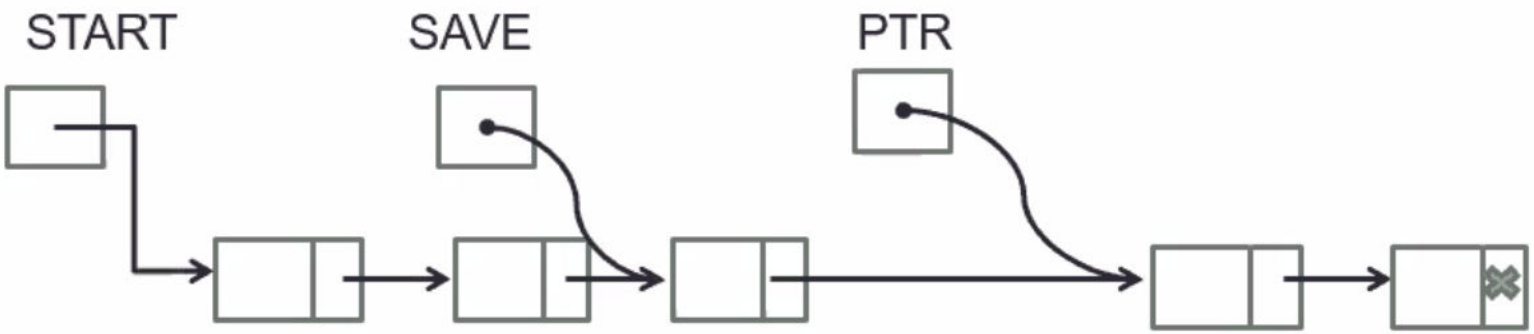
1. [OVERFLOW?] If avail=NULL then write: OVERFLOW and Exit.
2. [Remove first node from the avail list]
   Set New:=avail and avail:=link[avail]
3. Set info[New]:=item [Copies new data into new node]
4. Repeat step 5 while ptr ≠NULL
5. Set ptr:=link[ptr]
6. ptr:=New [New now becomes the last element]
7. link[ptr]:=NULL or link[New]:=NULL [Since New is the last elements its Next Pointer should points to the NULL pointer]
8. Exit.

Inserting at the End



Inserting into a Sorted List

- The suitable position for the item is found out.
- For this reason, two pointer need to be hold temporarily:
- One is for loc after which the item will be inserted.
- Other is for link[loc] which is the target address after item.



Inserting into a Sorted List

INSSRT(info, link, start, avail, item, loc)

This algorithm inserts item into a sorted linked list.

1. If start=NULL then: Set loc:=NULL
   Else if item<info[start] Set loc:=NULL
   Else: (a) Set save:=start and ptr:=link[start]
   (b) Repeat step (c) while ptr≠NULL and item>=info[ptr]
   (c) Set save:=ptr and ptr:=link[ptr]
   (d) Set loc:=save
2. If avail=NULL then Set write:OVERFLOW and Exit.
3. Set New:=avail and avail:=link[avail]
4. Set info[New]:=item
5. If loc=NULL then: Set link[New]:=start and start:=New //as first node
   Else: link[New]:=link[loc] and link[loc]:=New // after node with loc
6. Exit

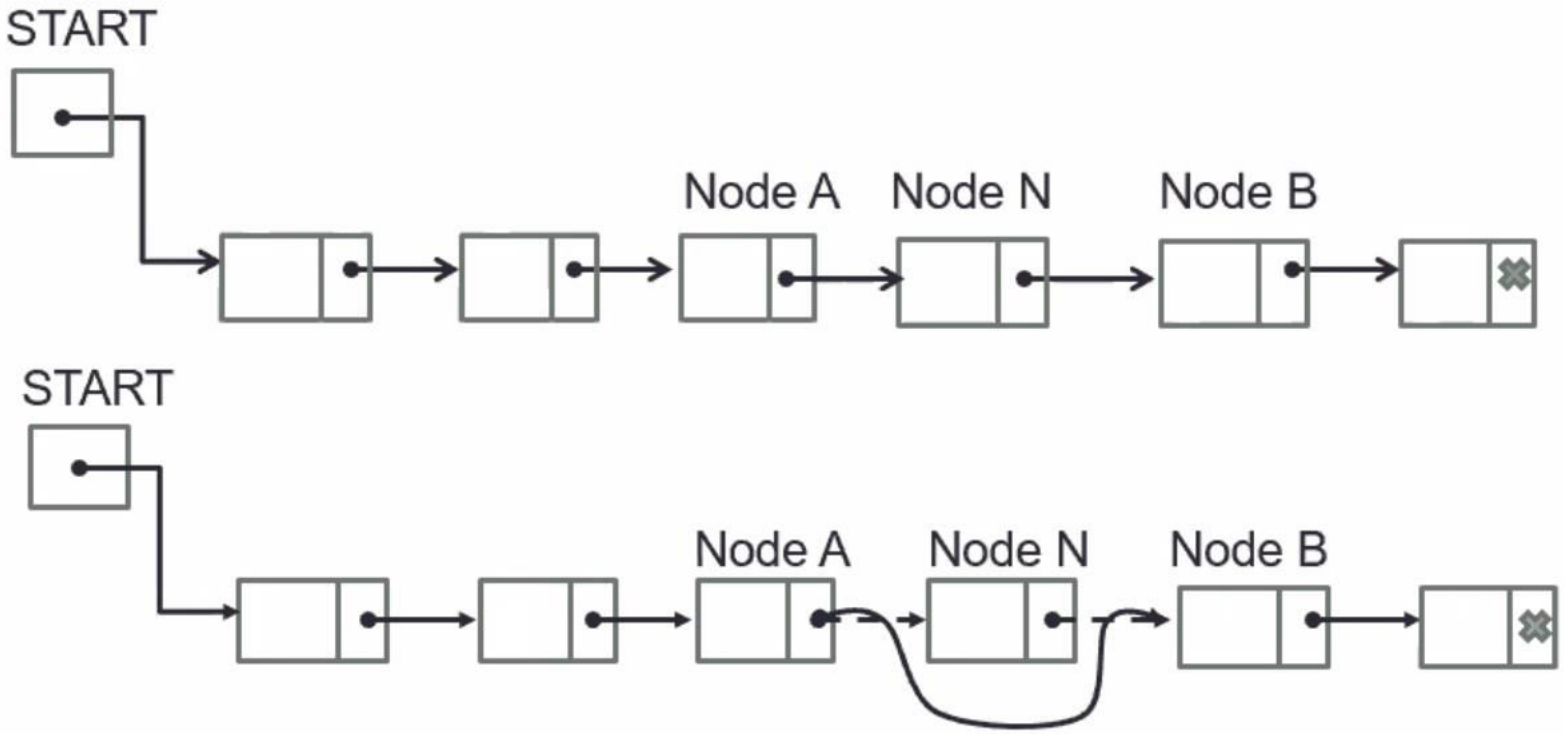
Deletion from a Linked List

- Deletion refers to remove a node from a list.
- Deletion can occur in the following situations:
  1. Deletion of a node with a given item of information.
  2. Deletion of the first node in the list.
  3. Deletion of a node following a given node in the list.
  4. Deletion of the last node in the list.

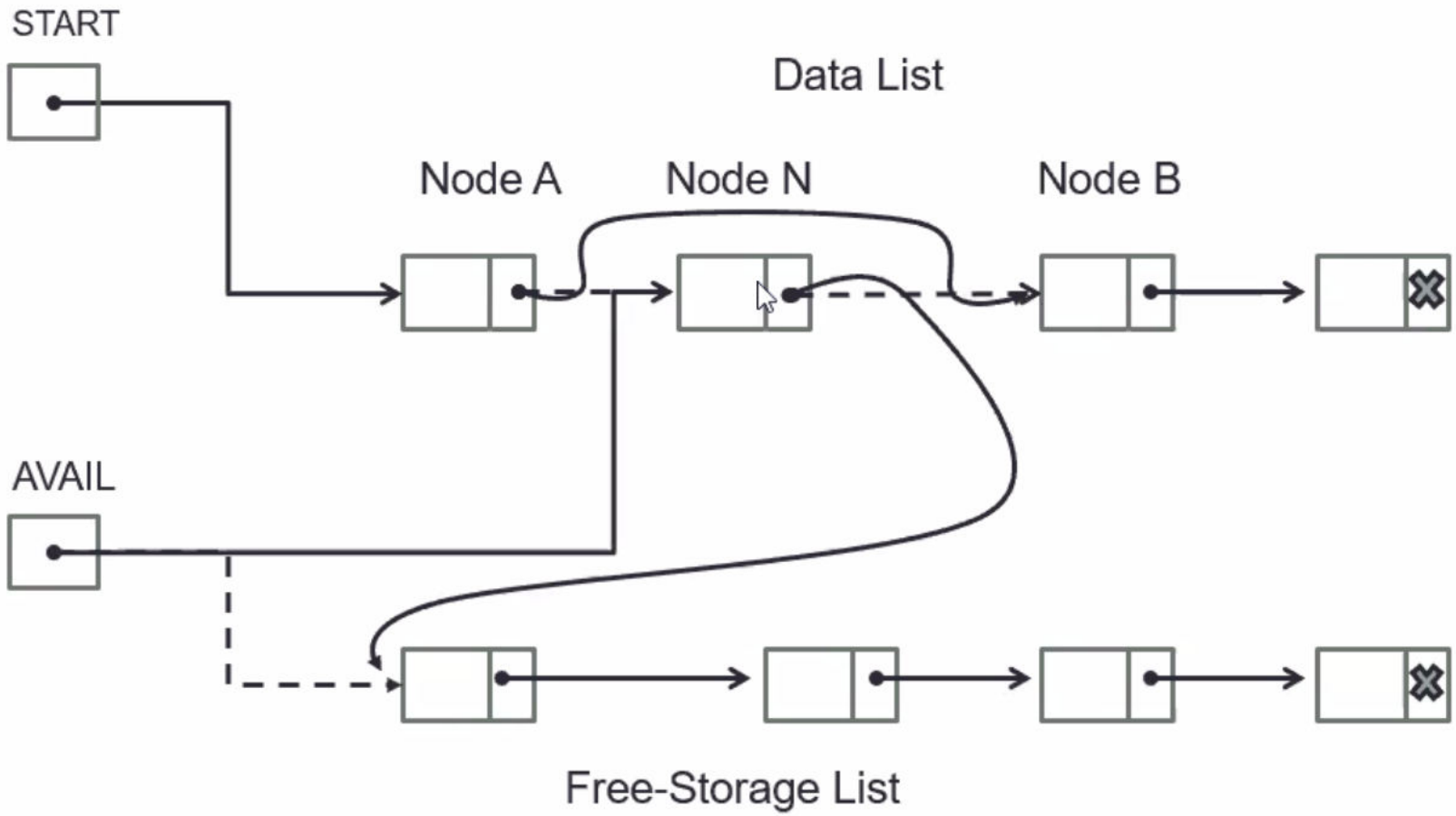
- When a node is deleted from a list, the deleted list should be added to the AVAIL list for future use by another list.
- How ever in C, we need only to free that node, the rest is done by the OS.

Deletion from a Linked List

- The Schematic diagram to delete node N which is between node A and B is shown below.



Deletion from a Linked List



Deletion from a Linked List

- During deletion, the deleted node is added to the free-storage list for future use by others.
- Three pointer fields are changed as follows:
  1. The nextpointer field of node A now points to node B where node N previously pointed.
  2. The nextpointer field of node N now points to the original first node in the free pool, where AVAIL previously pointed.
  3. AVAIL now points to the deleted node N.
- There are also two special cases:
  1. If the deleted node N is the first node in the list, then START will point to B.
  2. If the deleted node N is the last node in the list, then A will contain the null pointer.

Deleting Node with a Given Item

DeleteInfo(info, link, start, item, avail)

This algorithm deletes a node with a given item of information. It first search for a node containing item and its location loc and previous node location locp then add loc to free storage list.

1. If start=NULL, then: [Empty List]
   Set loc:=NULL and locp:=NULL
2. Else If info[start]=item, then: [First Node]
   Set loc:=start and locp:=NULL
3. Else:
   a) Set save:=start and ptr:=link[start] [Initialize Pointer]
   b) Repeat steps (c) and (d) while ptr≠NULL
   c) If info[ptr]=item, then Set loc:=ptr and locp:=save Break Loop.
   d) Save:=ptr and ptr:=link[ptr] [Updates Pointer]

Deleting Node with a Given Item Continue…

4. If info[ptr] ≠item, then: [If item is not in the List]
   Set loc:=NULL.
5. If loc=NULL, then:
   write: Item not in the list and Exit.
6. If locp=NULL, then: [Delete First Node]
   Set start:=link[start]
7. Else [Delete node with loc]
   Set link[locp]:=link[loc]
8. Set link[loc]:=avail and avail:=loc [Add deleted node to avail]
9. Exit

Deleting the Node Following a Given node (First, Middle or Last element)

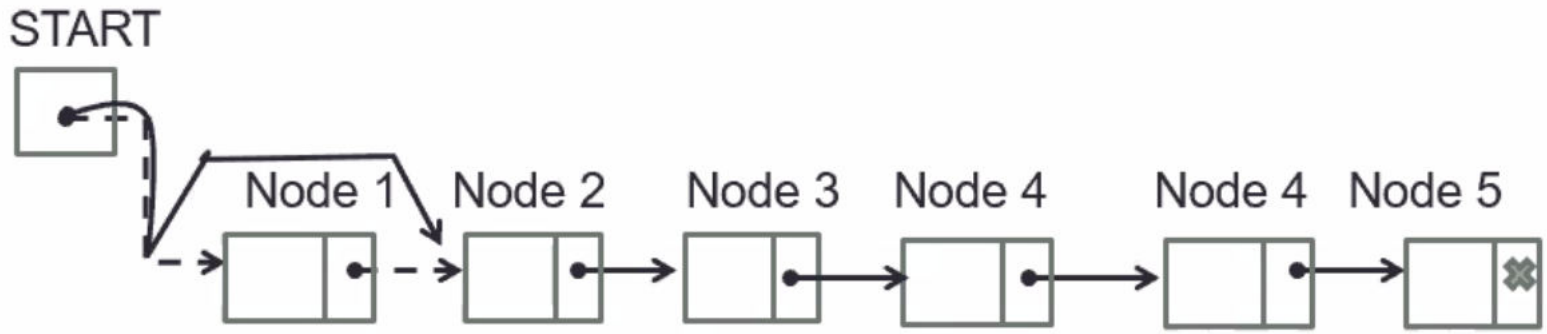
DEL(info, link, start, avail, loc, locp)

This algorithm deletes a node N with location loc. Locp is the location of the node which precedes N or when N is the first node, locp=NULL.

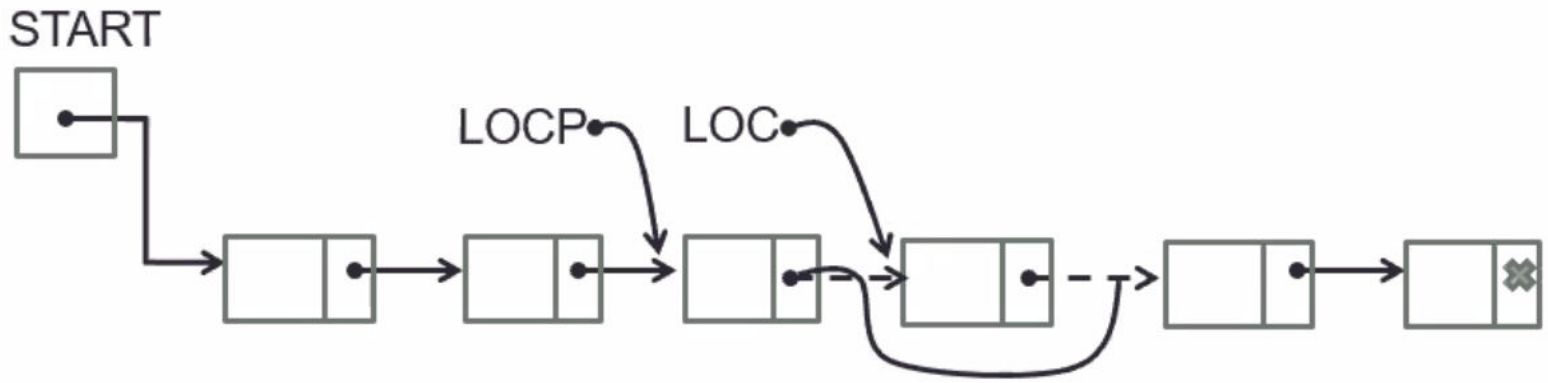
1. If locp=NULL, then: [Delete first node]
   Set start:=link[start]
2. Else [Delete node loc following locp]
   Set link[locp]:=link[loc]
3. Set link[loc]:=avail and avail:=loc
4. Exit

Deleting the Node Following a Given node (First, Middle or Last element)

Start:=link[Start]



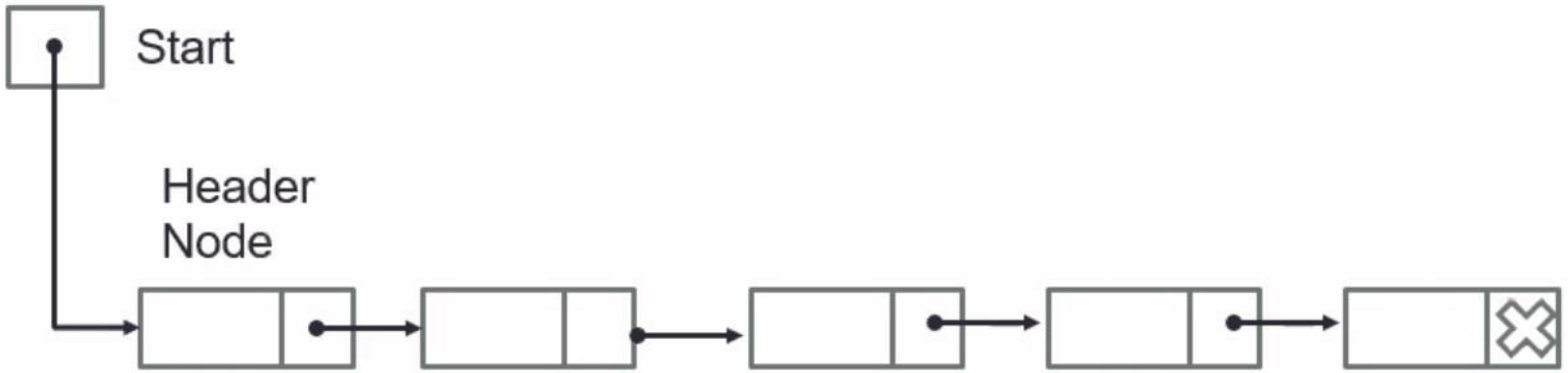
Link[locp]:=link[loc]



Header Linked List

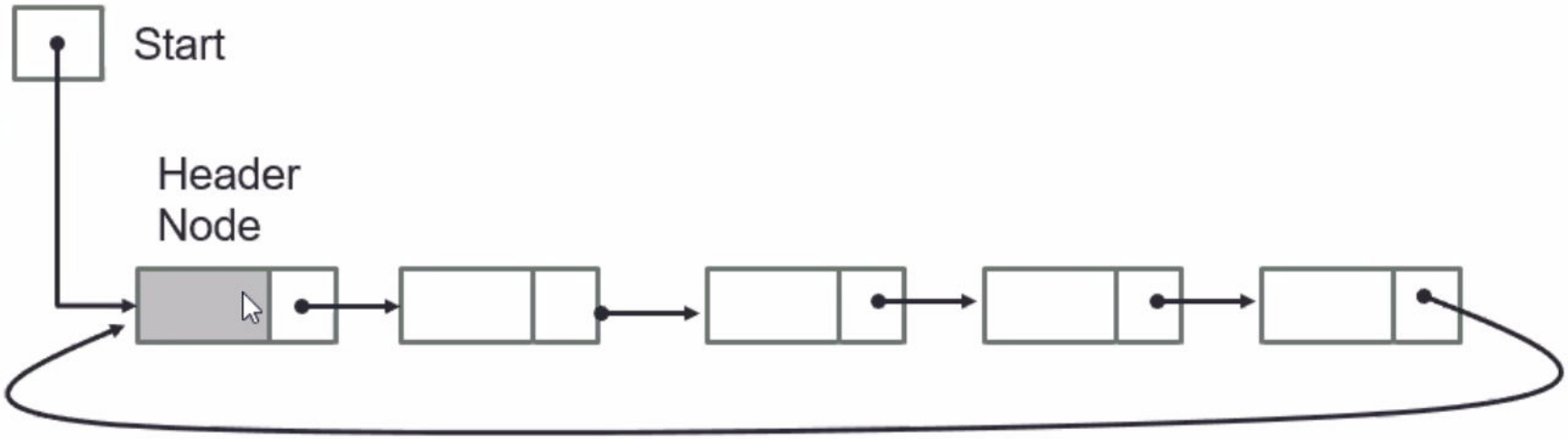
A header linked list is linked list which always contains a special node, called the header node, at the beginning of the list. Two kinds of header lists.

1. A Grounded header list: is a header list where the last node contains the null pointer.



Header Linked List

2. A Circular header list: is a header list where the last node points back to the header node.



Traversing a Circular Header List

Let list be a circular header list in memory. This algorithm traverse list by applying an operation Process to each node of list.

1. Set ptr:=link[start].
2. Repeat steps 3 and 4 while ptr≠start.
3. Apply process to info[ptr].
4. Set ptr:=link[ptr].
5. Exit.

Searching a Circular Header list

SRCHHL(info, link, start, item, loc)

This algorithm finds the location loc of the node where item first appears in list or sets loc=NULL.

1. Set ptr:=link[start].
2. Repeat while info[ptr]≠item and ptr≠start:
   Set ptr:=link[ptr]
3. If info[ptr]=item then:
   Set loc:=ptr
   Else: Set loc:=NULL
4. Exit

Deleting from a Circular Header List

This algorithm along with the following procedure deletes the first node N which contains item.

DELLOCHL(info, link, start, avail, item)

1. Call procedure FINDBHL(info, link, start, item, loc, locp) to find the location of N and its preceding node.
2. If loc=NULL then write: item not in list, and exit.
3. Set link[locp]:=link[loc].
4. Set link[loc]:=Avail and Avail:=loc.
5. Exit.

Deleting from a Circular Header List continue….

This procedure finds the location loc of the first node N which contains item and also the location locp of the node preceding N.

Procedure: FINDBHL(info, link, start, item, loc, locp)

1. Set save:=start and ptr:=link[start]
2. Repeat while info[ptr]≠item and ptr≠start
   Set save:=ptr and ptr:=link[ptr]
3. if info[ptr]=item then:
   Set loc:=ptr and locp=save
   Else: Set loc:=NULL and locp:=Save.
4. Exit.

Two-Way List

A two-way list is a linear collection of data elements, called node, where each node n is divided into three parts:

1. An information field INFO which contains the data of N.
2. A pointer field FORW which contains the location of the next node in the list.
3. A pointer field BACK which contains the location of the preceding node in the list.

This list also requires two list pointer variables: FIRST, which points to the first node in the list, and LAST, which points to the last mode in the list.

Two-Way List Schematic Diagram