Data Structure (CS-303)

Single Linked List

LINKED LIST

Why Linked List?

- Arrays can be used to store linear data of similar types, but arrays have the following limitations.
- 1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
- 2) Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.

LINKED LIST

Key Differences Between Array and Linked List

1. An array is the data structure that contains a collection of similar type data elements whereas the Linked list is considered as non-primitive data structure contains a collection of unordered linked elements known as nodes.

2. In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket.

3. In a linked list though, you have to start from the head and work your way through until you get to the fourth element.

LINKED LIST

Key Differences Between Array and Linked List

4. Accessing an element in an array is fast, while Linked list takes linear time, so it is quite a bit slower.

5. Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in Linked lists is fast.

6. Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expand and contract its size.

7. In an array, memory is assigned during compile time while in a Linked list it is allocated during execution or runtime.

LINKED LIST

Key Differences Between Array and Linked List

8. Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.

9. The requirement of memory is less due to actual data being stored within the index in the array. As against, there is a need for more memory in Linked Lists due to storage of additional next and previous referencing elements.

10. In addition memory utilization is inefficient in the array. Conversely, memory utilization is efficient in the linked list.

LINKED LIST

The Linked Lists advantages over Array

(1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, the upper limit is rarely reached.

(2) Inserting a new element in an array of elements is expensive because a room has to be created for the new elements and to create room existing elements have to be shifted.

LINKED LIST

Advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

LINKED LIST

Drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation. Read about it here.

2) Extra memory space for a pointer is required with each element of the list.

3) Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

LINKED LIST

Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.

Each node in a list consists of at least two parts:

1) data

2) Pointer (Or Reference) to the next node

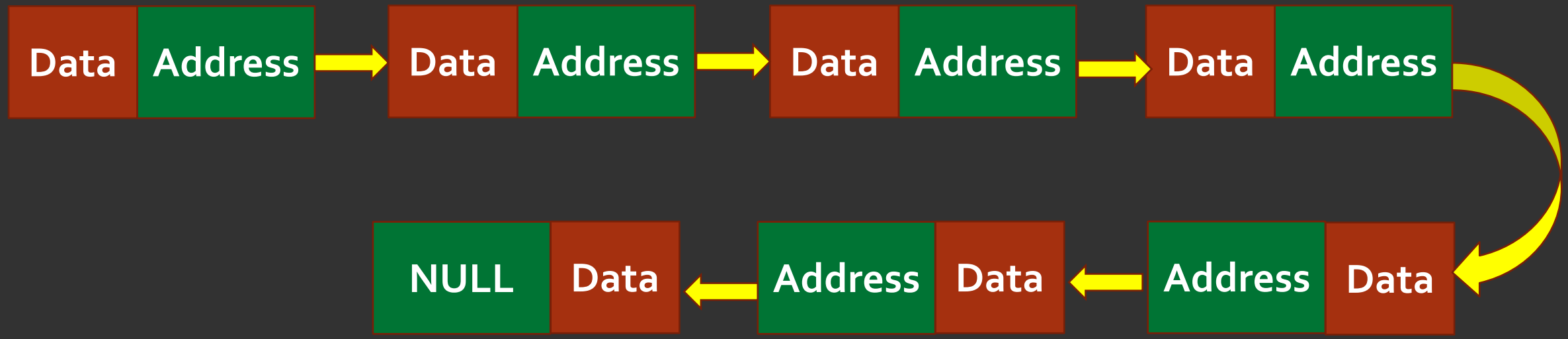
LINKED LIST

```
// A linked list node
struct Node {
        int data;
        struct Node* next;
};
```

LINKED LIST

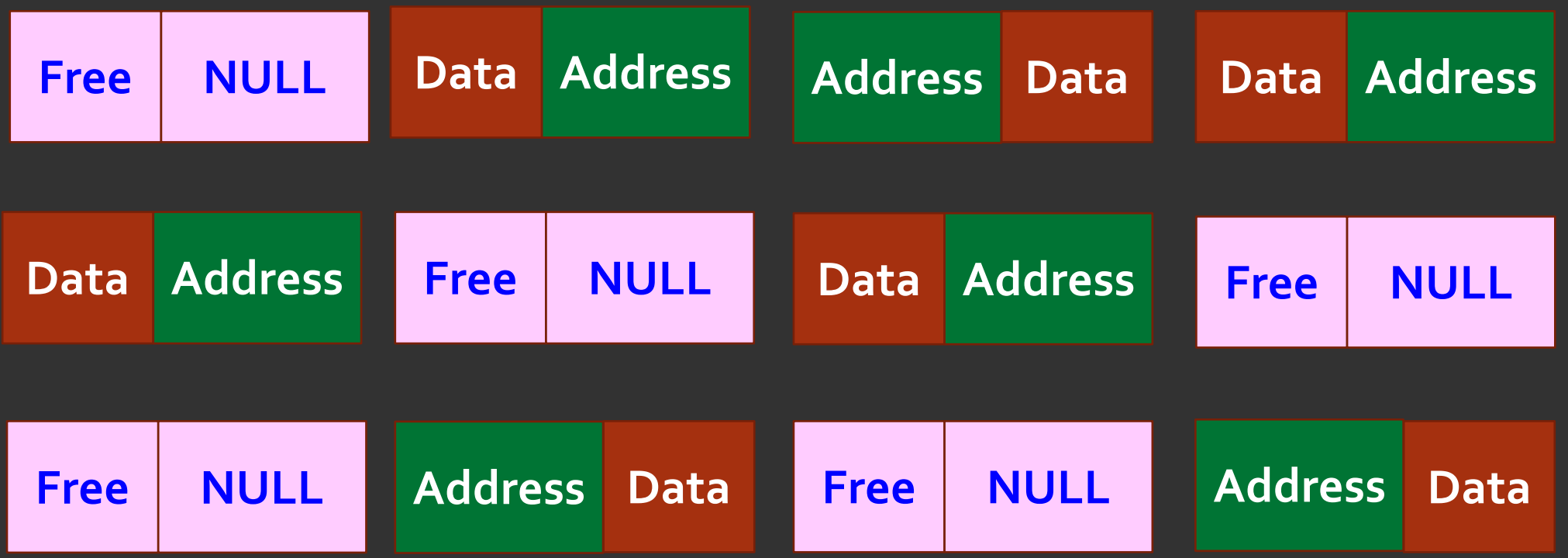
```
class Node {
public:
        int data;
        Node* next;
};
```

A TYPICAL SINGLY LINKED LIST

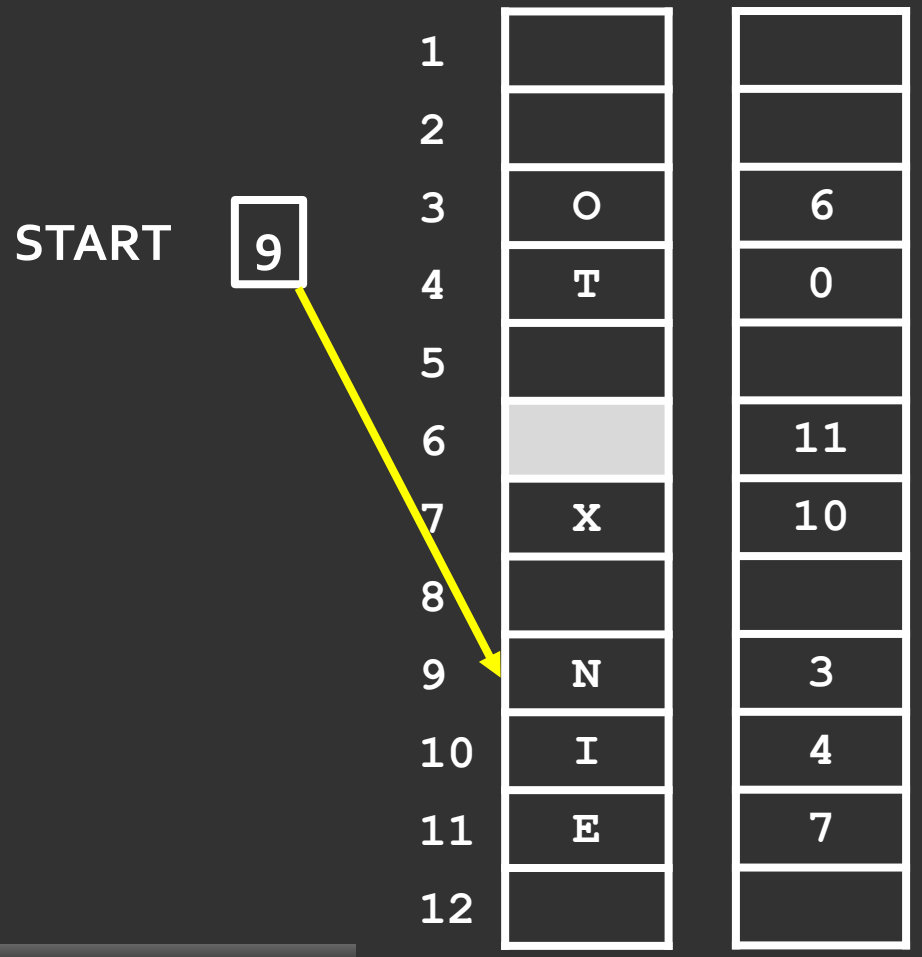


A TYPICAL SINGLY LINKED LIST

A TYPICAL SINGLY LINKED LIST IN MEMORY



A TYPICAL SINGLY LINKED LIST IN MEMORY



TRAVERSAL IN LINKED LIST

1. Set PTR := START
2. Repeat Steps 3 and while PTR ≠ NULL
3. Apply PROCESS to INFO[PTR]
4. Set PTR := LINK [PTR]

   End of Step 2 Loop
5. Exit

PRINT INFORMATION OF LINKED LIST

PRINT (INFO, LINK, START)

1. Set PTR:=START
2. Repeat Steps 3 and 4 while PTR ≠NULL:
3. Write: INFO[PTR]
4. Set PTR := LINK[PTR]

   End of Step 2 loop
5. Return

SEARCHING A LINKED LIST

SEARCH (INFO, LINK, START, ITEM, LOC)

1. Set PTR:=START
2. Repeat Step 3 while PTR ≠ NULL:
3. If ITEM = INFO[PTR], then :
   Set LOC := PTR, and Exit [ITEM found]
   Else:
   Set PTR := LINK[PTR]
   [End of If structure]
   [End of Step 2 Loop]
4. Set LOC := NULL [Search is unsuccessful]
5. Exit

SEARCHING A LINKED LIST (SORTED LIST)

SEARCH (INFO, LINK, START, ITEM, LOC)

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 [ITEM found]
   Else:
   Set LOC := NULL, and Exit [ITEM now exceeds INFO[PTR]]
   [End of If structure]
   [End of Step 2 Loop]
4. Set LOC := NULL [Search is unsuccessful]
5. Exit

SOME CONCEPTS

- AVAIL LIST
- GARBAGE COLLECTION
- OVERFLOW
- UNDERFLOW

INSERTION IN LINKED LIST (AT BEGINNING)

INSFIRST (INFO, LINK, START, AVAIL, ITEM)

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

INSERTION IN LINKED LIST (AFTER GIVEN NODE)

INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)

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. If LOC=NULL, then: [Insert as first node]

   Set LINK[NEW] := START and Set START := NEW

   Else: [Insert after node with location LOC]

   Set LINK[NEW] := LINK[LOC] and LINK[LOC] := NEW

   [End of If structure]
5. Exit

INSERTION IN LINKED LIST (IN A SORTED LIST)

FINDA (INFO, LINK, START, AVAIL, ITEM)

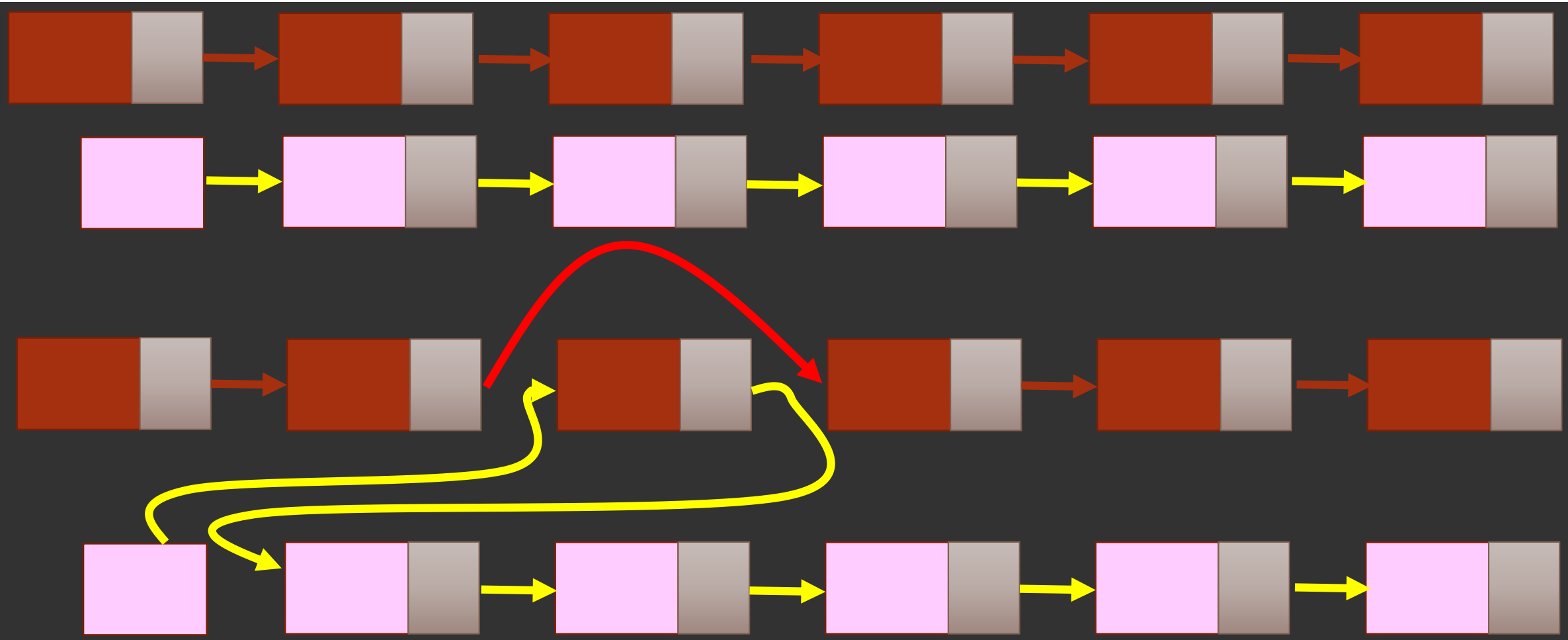
1. [List empty?] If START=NULL, then:
   Set LOC := NULL, and return
2. [Special Case?] If ITEM<INFO[START], then:
   Set LOC := NULL, and return
3. Set SAVE := START and PTR := LINK [START] [Initialized pointers]
4. Repeat Steps 5 and 6 while PTR ≠ NULL
5. If ITEM < INFO[PTR], then:
   Set LOC := SAVE, and return
   [End of If structure]
6. Set SAVE := PTR and PTR := LINK[PTR] [Updates pointers]
   [End of Step 4 loop]
7. Set LOC := SAVE
8. Return

INSERTION IN LINKED LIST (IN A SORTED LIST)

INSSORTED (INFO, LINK, START, AVAIL, ITEM)

1. Call FINDA (INFO, LINK, START, ITEM, LOC)
2. Call INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)
3. Exit

DELETION IN LINKED LIST



DELETING THE NODE FOLLOWING A GIVEN NODE

DEL (INFO, LINK, START, AVAIL, LOC, LOCP)

1. If LOCP=NULL, then:

   Set START := LINK [START] [Deletes first node]

   Else:

   Set LINK [LOCP] := LINK [LOC] [Deletes node N]

   [End of If structure]

2. [Return deleted node to the AVAIL list]

   Set LINK [LOC] := AVAIL and AVAIL:= LOC

3. Exit

DELETION OF NODE WITH A GIVEN ITEM OF INFORMATION

- What this algorithm will do?
  - Delete the first node with given Information value.
- Requirements
  - We must know the location of node which contains the given information.

DELETION OF NODE WITH A GIVEN ITEM OF INFORMATION

FINDB(INFO, LINK, START, ITEM, LOC, LOCP)

[It will find the location LOC of the first node N which contains ITEM and location LOCP of the node preceding N. If ITEM does not appear in the list, then the procedure sets LOC=NULL, and if ITEM appears in the first node, then it sets LOCP=NULL]

1. If START=NULL, then: [List empty?]
   Set LOC := NULL and LOCP :=NULL, and Return. [End of IF Structure]
2. If INFO[START]=ITEM, then: [ITEM in first node?]
   Set LOC := START and LOCP = NULL, and Return [End of IF Structure]
3. Set SAVE:=START and PTR:= LINK[START] [Initialized Pointers]
4. Repeat Steps 5 and 6 while PTR ≠ NULL
5. If INFO[PTR] = ITEM, then:
   Set LOC := PTR and LOCP := SAVE, and Return [End of IF Structure]
6. Set SAVE := PTR and PTR := LINK[PTR] [Updates pointers]

   [End of Step 3 Loop]
7. Set LOC := NULL [Search unsuccessful]
8. Return

DELETION OF NODE WITH A GIVEN ITEM OF INFORMATION

DELETE (INFO, LINK, START, AVAIL, ITEM)

1. Call FINDB (INFO, LINK, START, ITEM, LOC, LOCP)
2. If LOC = NULL, then: Write: ITEM not in list, and Exit
3. If LOCP = NULL, then: [Delete Node]
   Set START := LINK[START] [Deletes first node]
   Else:
   Set LINK [LOCP] := LINK [LOC]
   [End of If structure]
4. Set LINK[LOC] := AVAIL and AVAIL := LOC
   [Return deleted node to the AVAIL LIST]
5. Exit

DELETION IN LINKED LIST

Delete First Node of Linked List

1. IF START = NULL
   Write UNDERFLOW
   Go to Step 5
   [END OF IF]
2. SET PTR = START
3. SET START = START NEXT
4. FREE PTR
5. EXIT

DELETION IN LINKED LIST

Delete Last Node of Linked List

1. IF START = NULL
   Write UNDERFLOW, and Go to Step 8
   [END OF IF]
2. SET PTR = START
3. Repeat Steps 4 and 5 while PTR NEXT != NULL
4. SET PREPTR = PTR
5. SET PTR = PTR NEXT
   [END OF LOOP]
6. SET PREPTR NEXT = NULL
7. FREE PTR
8. EXIT