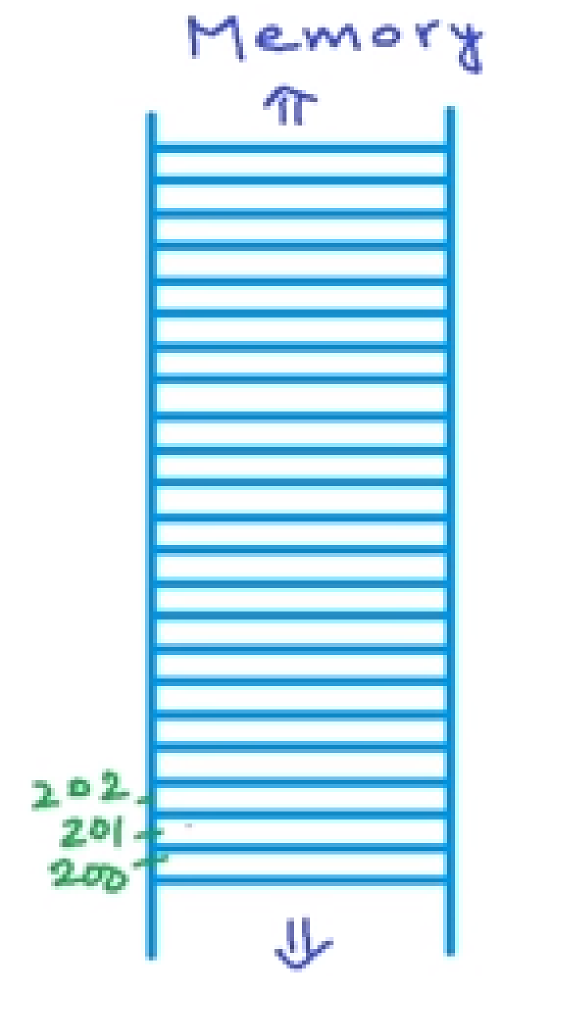
Data Structures and Algorithms

Shikha Mehrotra

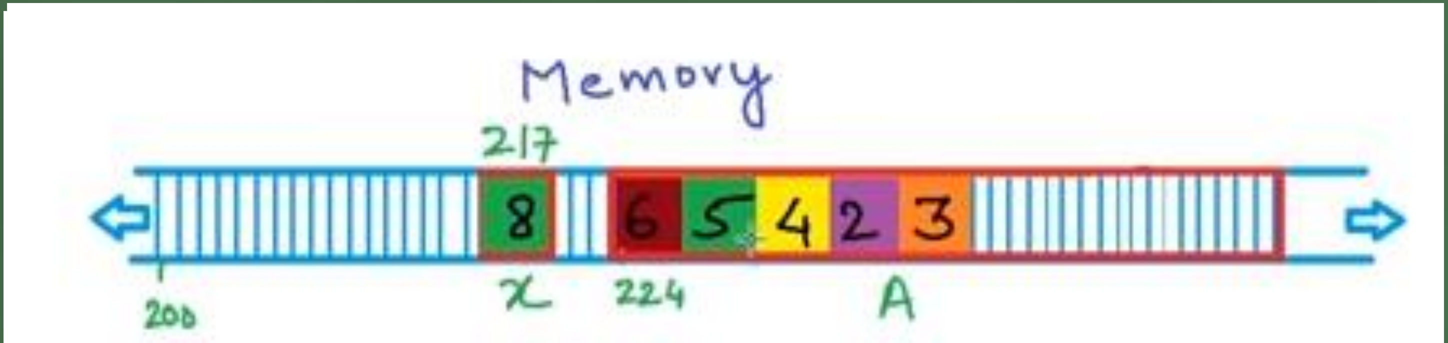
Linked List

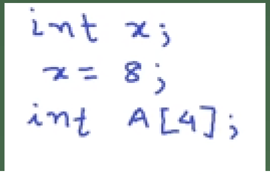
Introduction of Linked list



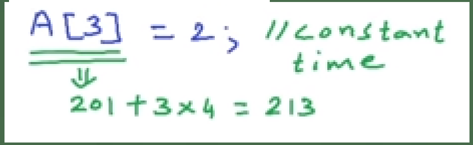
Introduction of Linked list

Albert





```
int x;
 x= 8;
int  A[4];
```

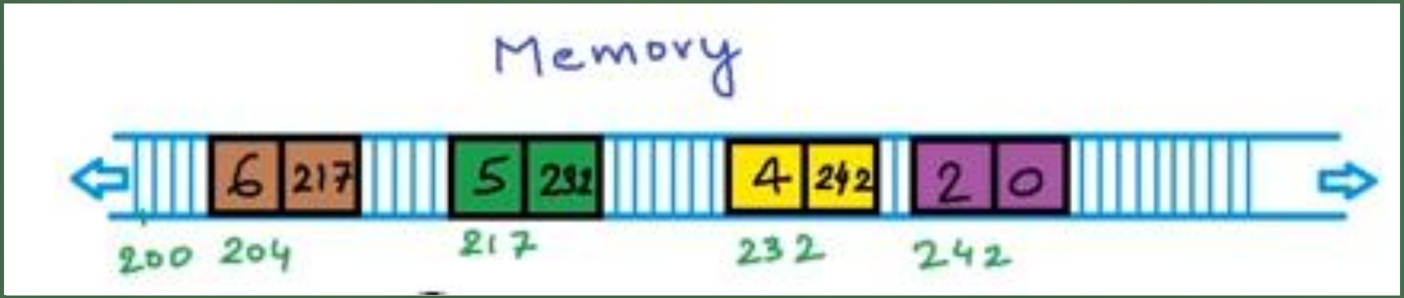


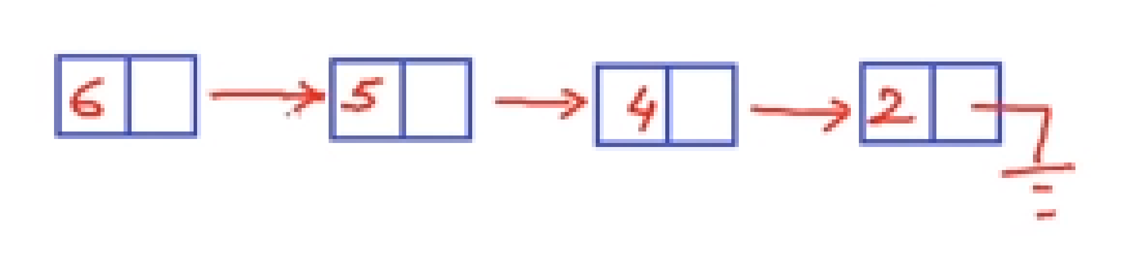
```
A[3]  = 2;  //constant
                time
  ⇓
   201 +3×4 = 213
```

Memory Manager

Introduction of Linked list

Albert





6, 5, 4, 2

```
Struct node
{
    int data;
    node* next;
}
```

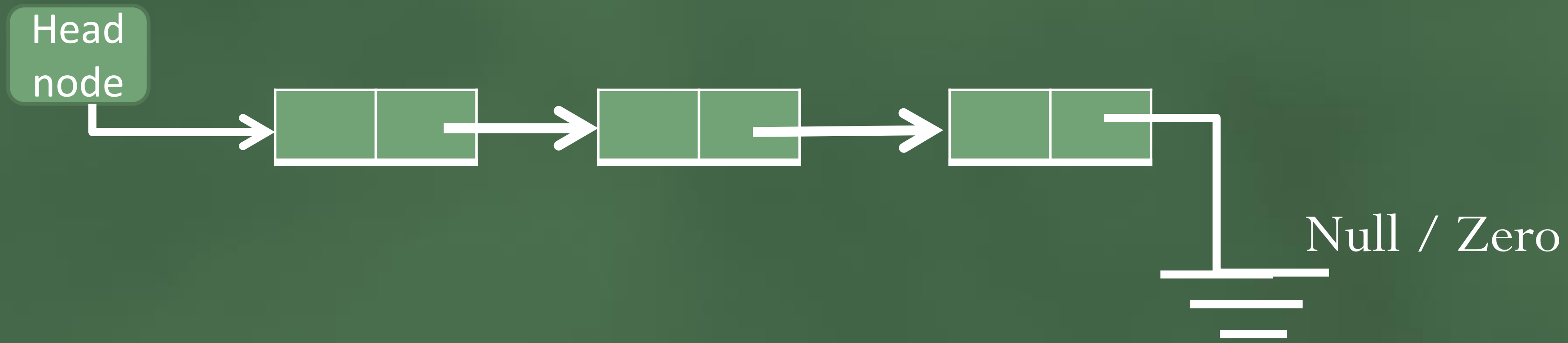
Memory Manager

•Limitation of Arrays

- An array has a limited number of elements
  - routines inserting a new value have to check that there is room
- Can partially solve this problem by reallocating the array as needed (how much memory to add?)
  - adding one element at a time could be costly
  - one approach - double the current size of the array
- A better approach: use a *Linked List*

•Anatomy of a linked list

- A linked list consists of:
  - A sequence of nodes



Each node contains a value
and a link (pointer or reference) to some other node

The last node contains a null link

The list must have a header

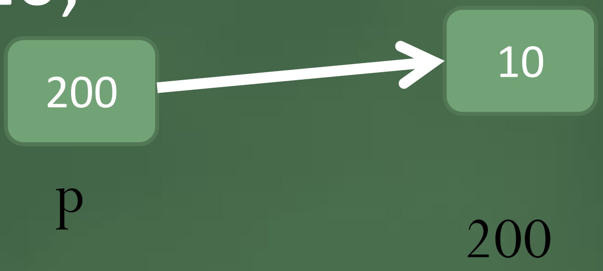
Value	link

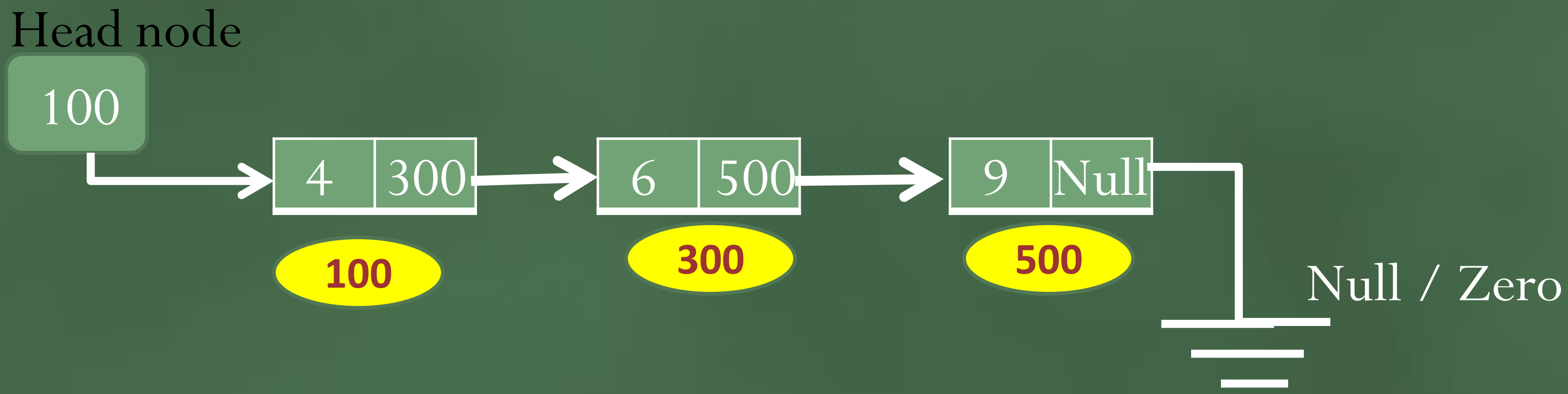
Terminology

- **Head (front, first node):**
  - The node without predecessor, the node that starts the lists.
- **Tail (end, last node):**
  - The node that has no successor, the last node in the list.
- **Current node:** The node being processed.
  - From the current node we can access the next node.
- **Empty list:** No nodes exist

- **More terminology**
  - A node’ s successor is the next node in the sequence
    - The last node has no successor
  - A node’ s predecessor is the previous node in the sequence
    - The first node has no predecessor
  - A list’ s length is the number of elements in it
    - A list may be empty (contain no elements)

- **pointers recap**
- Int *p;
- P = (int *)malloc(sizeof(int));
- *p=10;





Head node

100

4 | 300

6 | 500

9 | Null

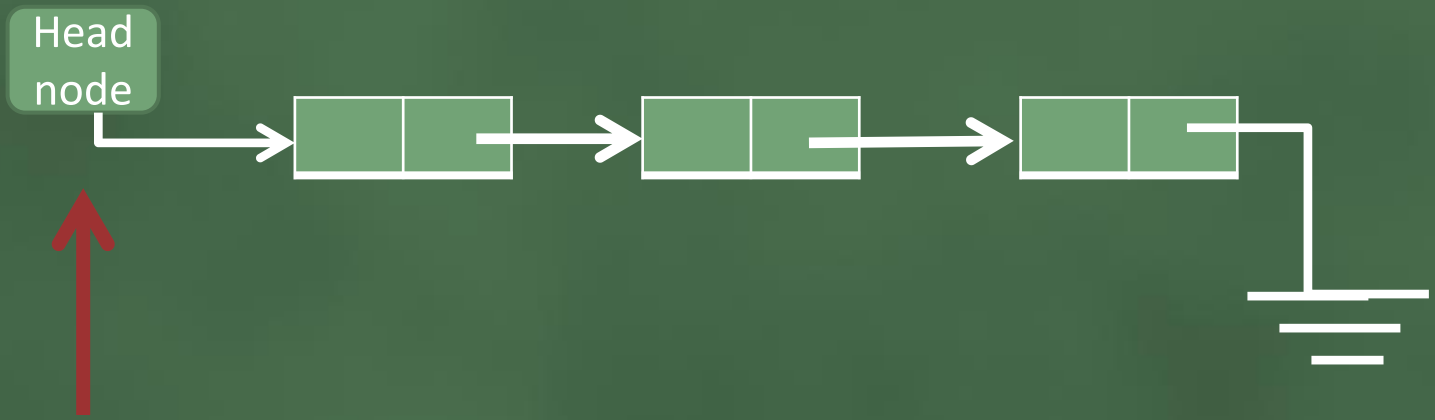
100

300

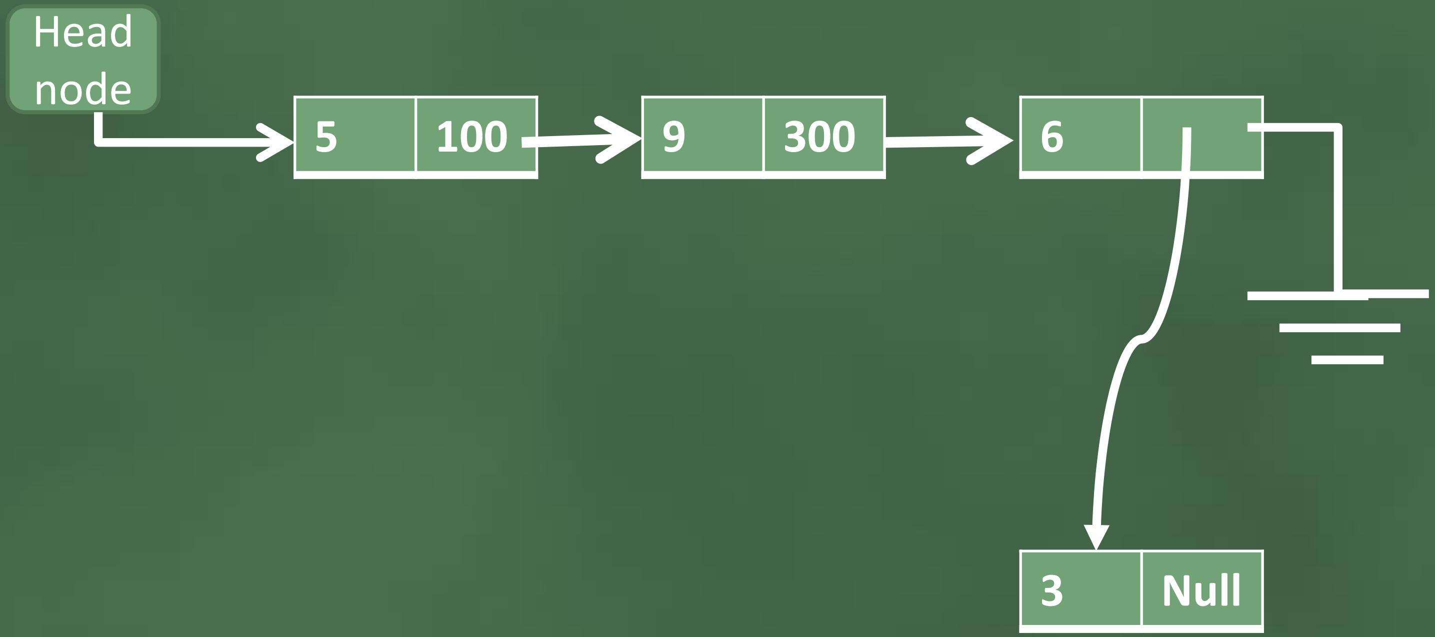
500

Null / Zero

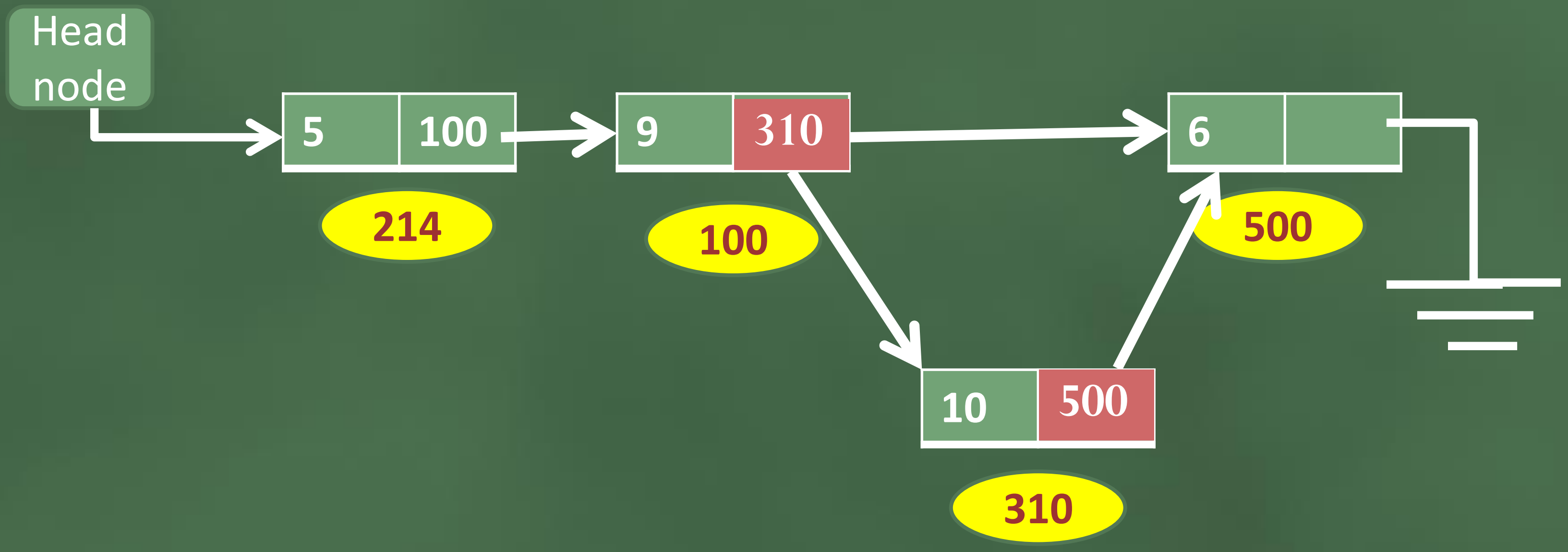
- **Traversal**



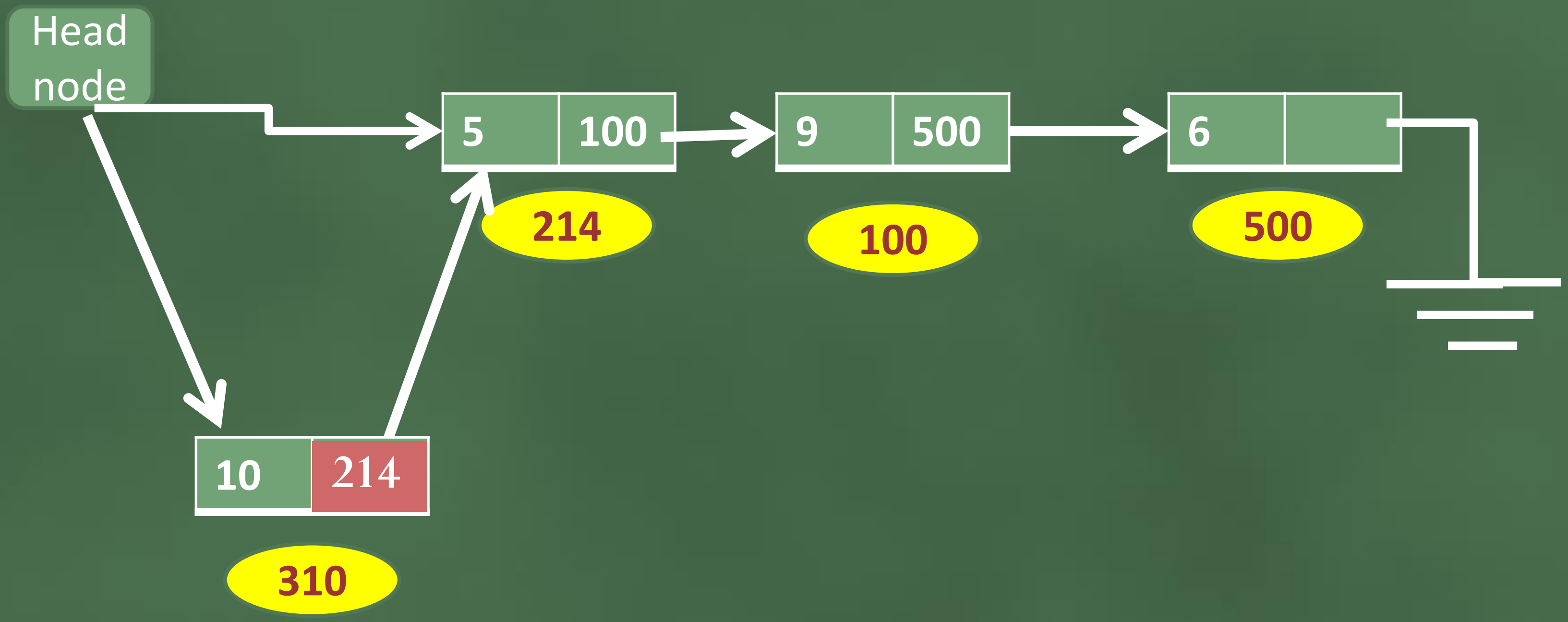
- **Insertion – at the end of list**



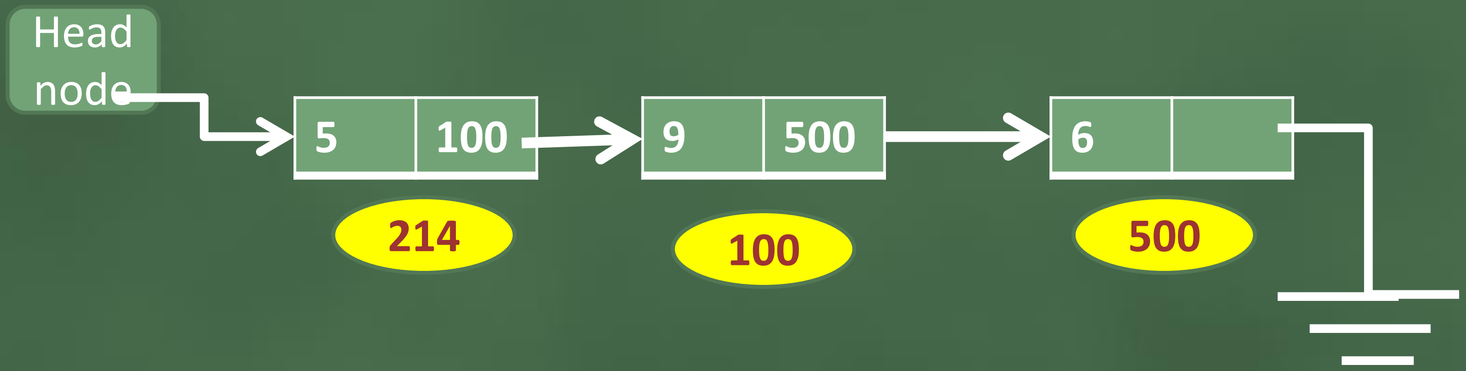
- Insertion – between two nodes



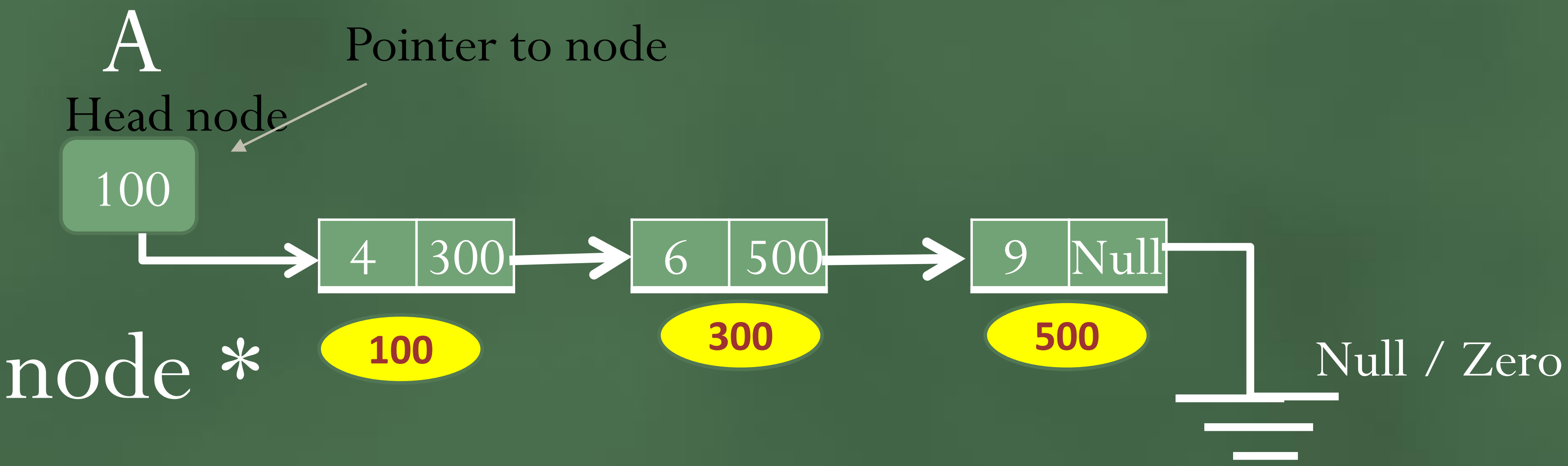
- **Insertion - beginning of linked list**



Deletion - last node



Implementation of linked list



A

Pointer to node

Head node

100

4 300

6 500

9 Null

100

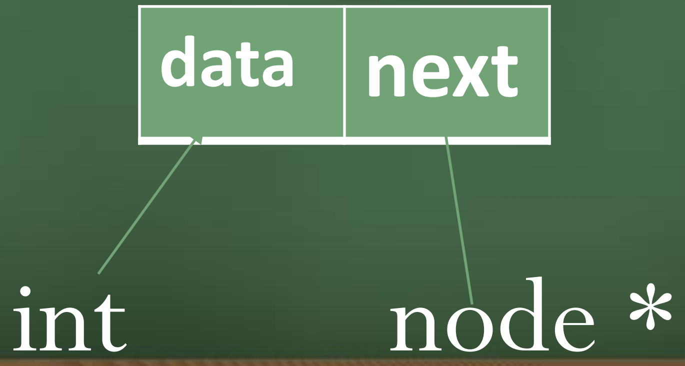
300

500

node *

Null / Zero

```
Struct node
{
    int data;
    node* next;
}
```



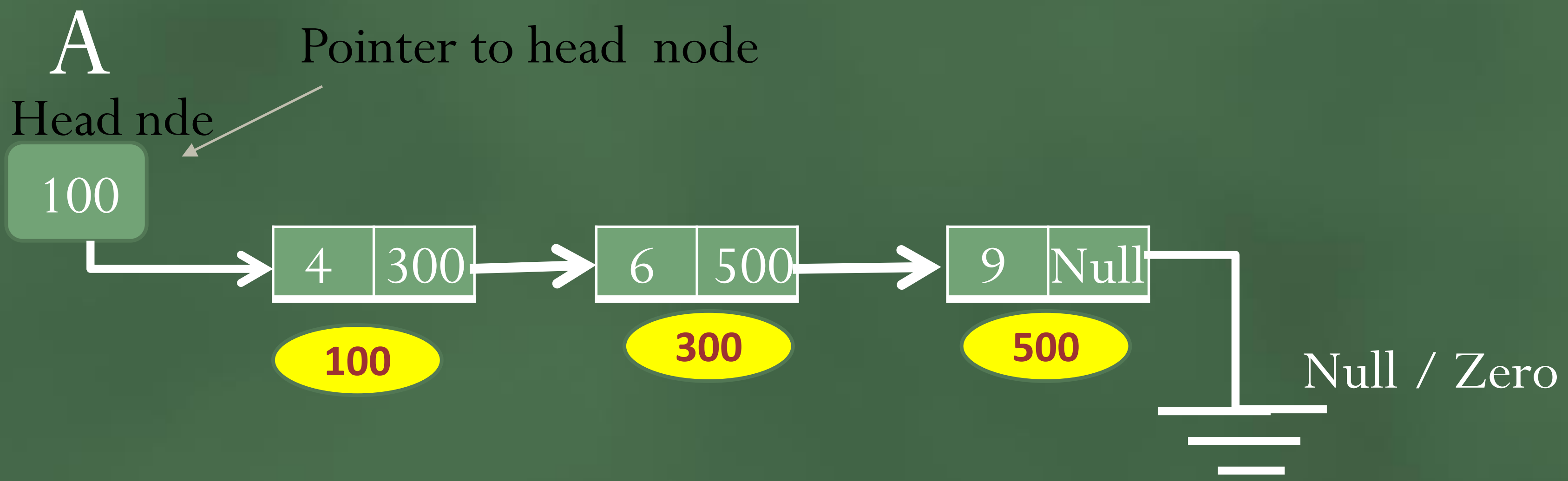
data

next

int

node *

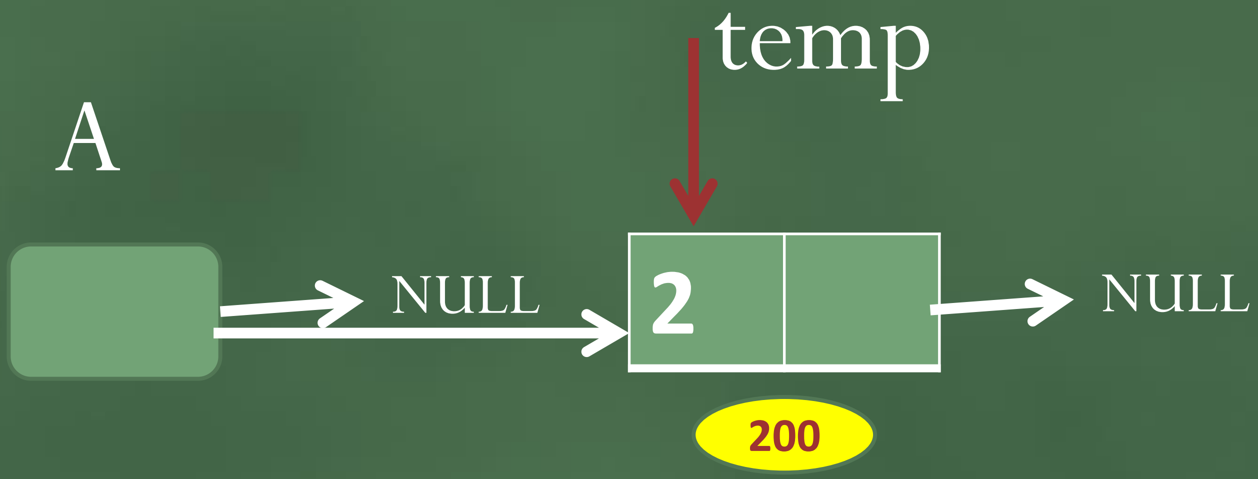
Implementation of linked list



```
Struct node
{
    int data;
    node* next;
}
```

```
Node* A
A = NULL
```

Implementation of linked list – inserting first node

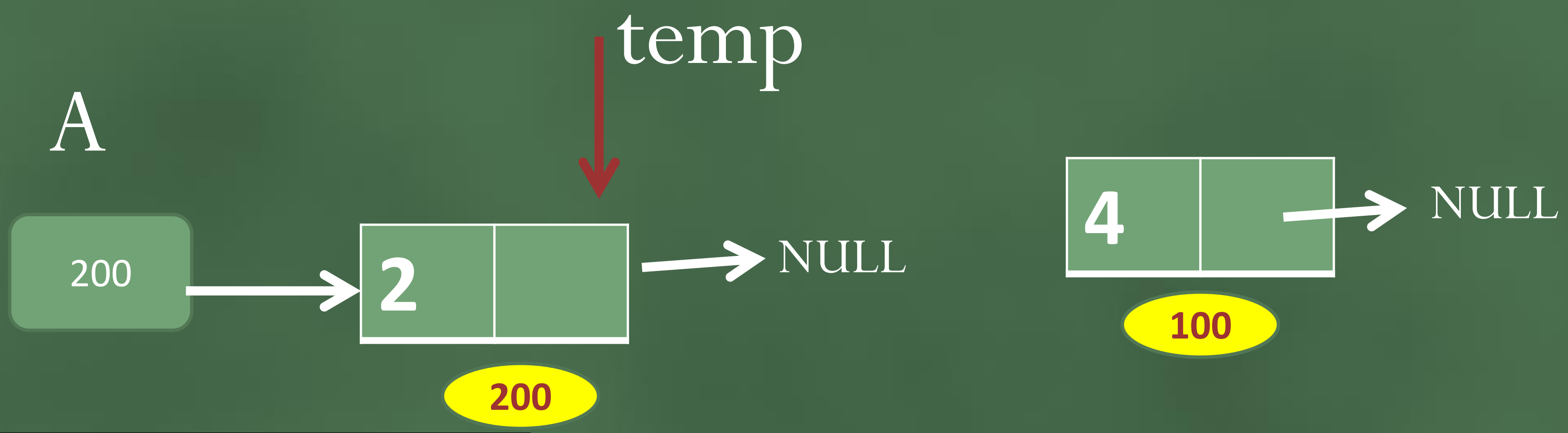


```
Struct node
{
    int data;
    node* next;
}
```

```
Node* A
A = NULL
```

```
Node* temp =(Node*)
malloc (sizeOf(Node)

 temp -> data = 2
 temp -> next = NULL

A=temp
```

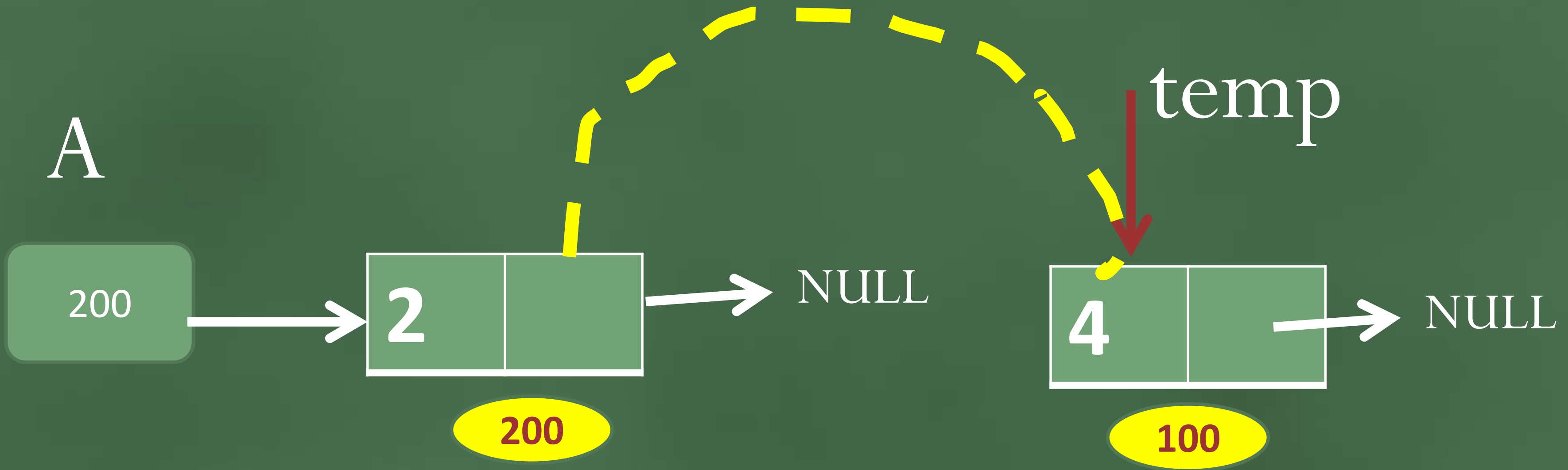
Insertion at the end



```
Node* temp =(Node*)
malloc (sizeOf(Node)

 temp -> data = 2

 temp -> next = NULL

A=temp
```

```
Temp = (node*)malloc(sizeOf(Node)

temp -> data = 4
temp -> next = NULL
```

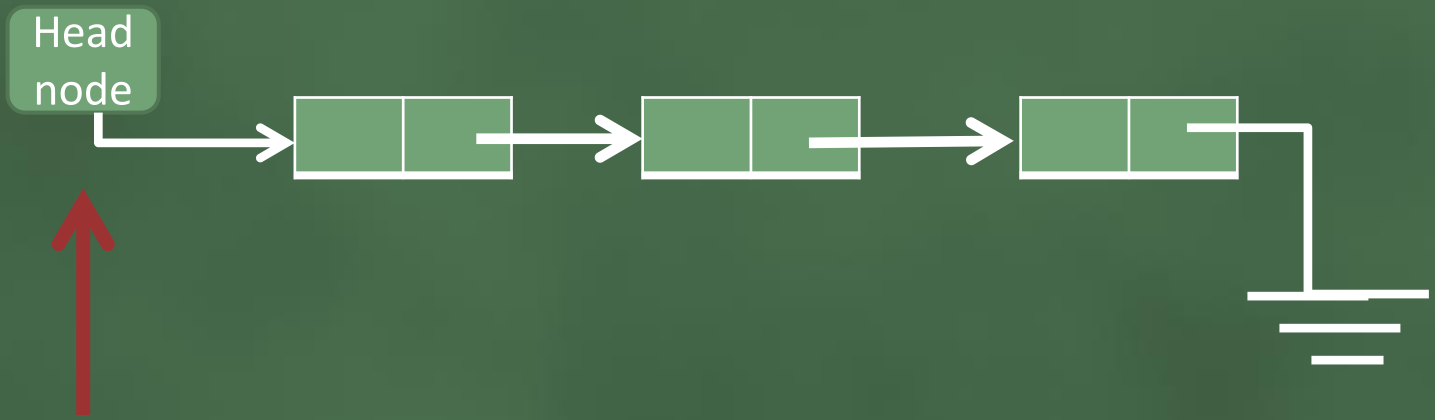
Traversal



```
Temp =
(node*)malloc(sizeOf(Node)

temp -> data = 4
temp -> next = NULL
```

```
Node* temp1 = A
While (temp1->next !=NULL)
{
        temp1=temp1->next
}
```

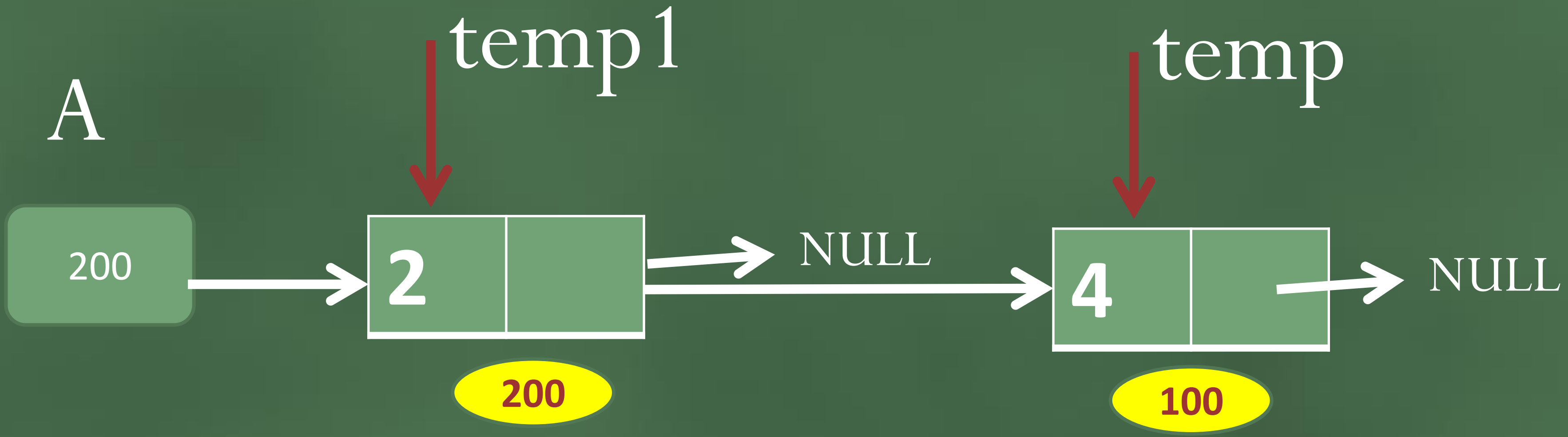
Traversal



```
Temp =
(node*)malloc(sizeOf(Node)

temp -> data = 4
temp -> next = NULL
```

```
Node* temp1 = A
While (temp1->next !=NULL)
{
        temp1=temp1->next
}
```

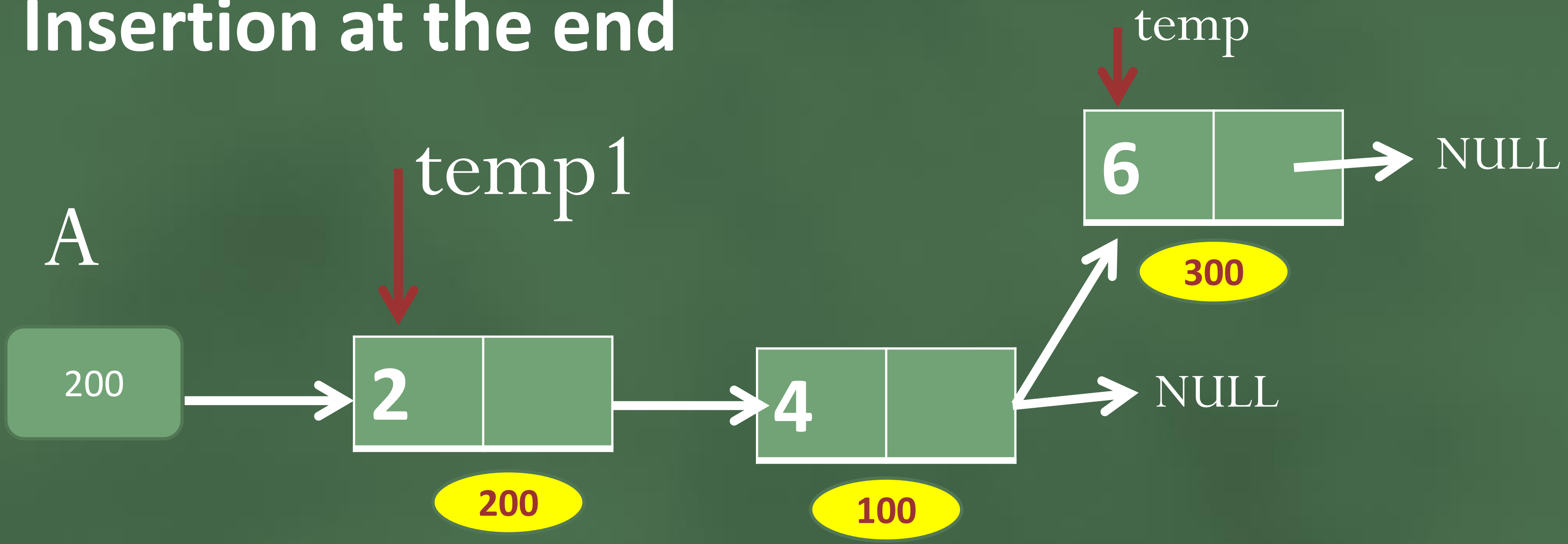
Insertion at the end



```
Temp =
(node*)malloc(sizeOf(Node)

temp -> data = 4
temp -> next = NULL
```

```
Node* temp1 = A
While (temp1->next !=NULL)
{
        temp1=temp1->next
}

Temp1-> next = temp
```

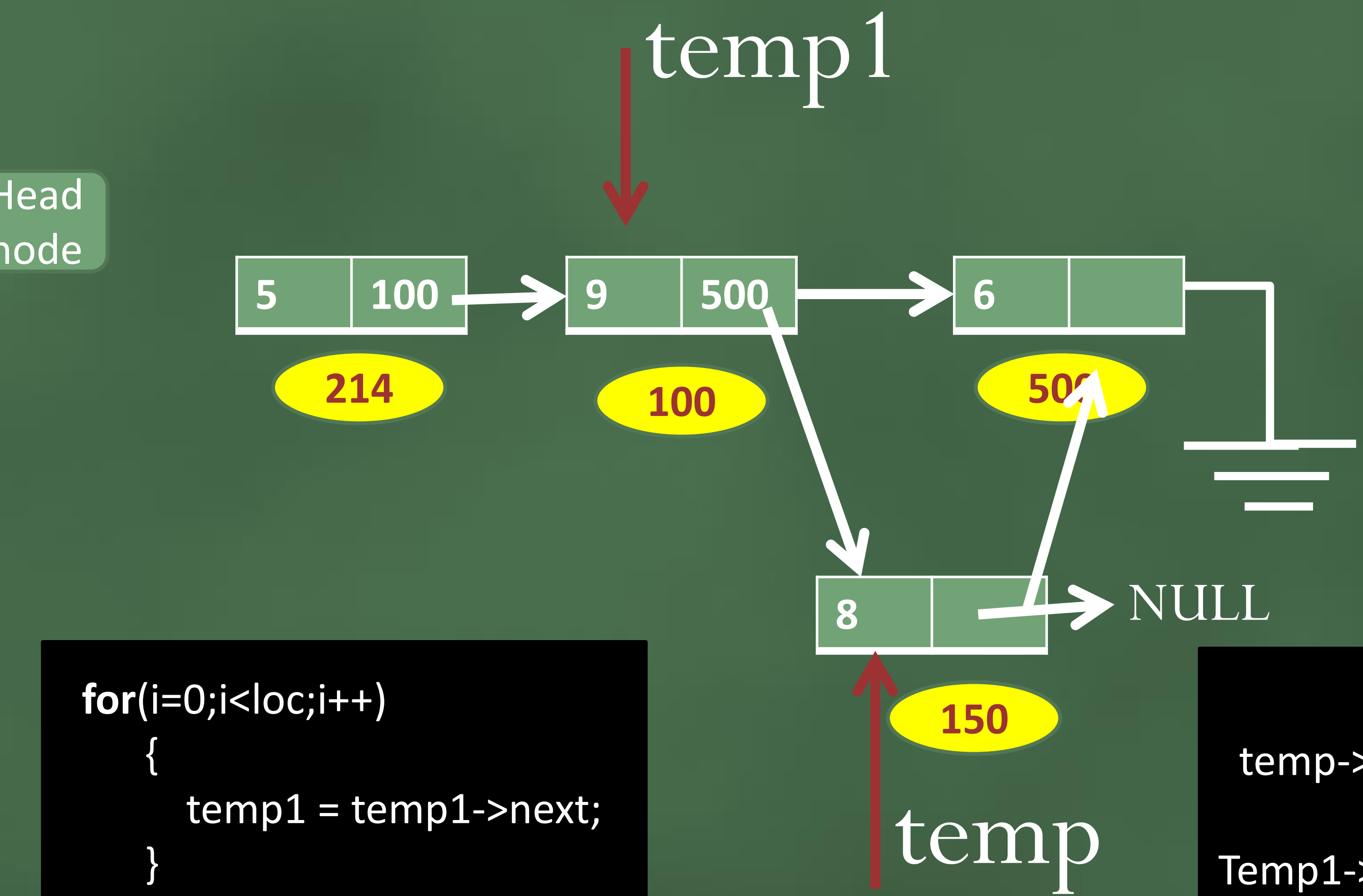
Insertion at the end



```
Temp =
(node*)malloc(sizeOf(Node)

temp -> data = 4
temp -> next = NULL
```

```
Node* temp1 = A
While (temp1->next !=NULL)
{
        temp1=temp1->next
}

Temp1-> next = temp
```

Insert node at nth position



```
for(i=0;i<loc;i++)
    {
       temp1 = temp1->next;
    }
```

```
temp->next = temp1 ->next;

Temp1->next= temp
```

```
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head, *ptr;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();

ptr = (struct node *)malloc(siz
Int main()
{
 int choice =0;
    while(choice != 9)
    {
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
eof(struct node *));
```

```
          {
              case 1:
                   beginsert();     break;
              case 2:
                   lastinsert();       break;
              case 3
                  randominsert();     break;
              case 4:
                   begin_delete();     break;
              case 5:
                last_delete();       break;
              case 6:
                 random_delete();       break;
              case 7:
                   search();       break;
              case 8:
                      display();     break;
              case 9:
                        exit(0);   break;
              default:
                  printf("Please enter valid choice.
          }
      }
  }
```

```
void beginsert()
{
   struct node *ptr;
   int item;
   ptr = (struct node *) malloc(sizeof(struct node *));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value\n");
      scanf("%d",&item);
      ptr->data = item;
      ptr->next = head;
      head = ptr;
      printf("\nNode inserted");
   }

}
```

```
void lastinsert()
{      struct node *ptr,*temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct
   if(ptr == NU {
      printf("\nOVERFLOW");  }
   else  {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL {
         ptr -> next = NULL;
         head = ptr;
         printf("\nNode inserted");
      }
      else  {
         temp = head;
         while (temp -> next != NULL) {
            temp = temp -> next;
         }
         temp->next = ptr;
         ptr->next = NULL;
         printf("\nNode inserted");
      }    }  }
```

```
void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want t
        scanf("\n%d",&loc);
        temp=head;
```

```
    for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }

        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
}
```