Chapter 5

Lists

Lists

- A list has sequence of elements with a fixed order
- We can add, remove, inspect, and update elements anywhere in a list; Hence, lists are more general than stacks or queues
- The **length** of the list is the number of elements it contains
- The empty list has length zero
- We can implement lists using both arrays and linked lists

Linked Lists

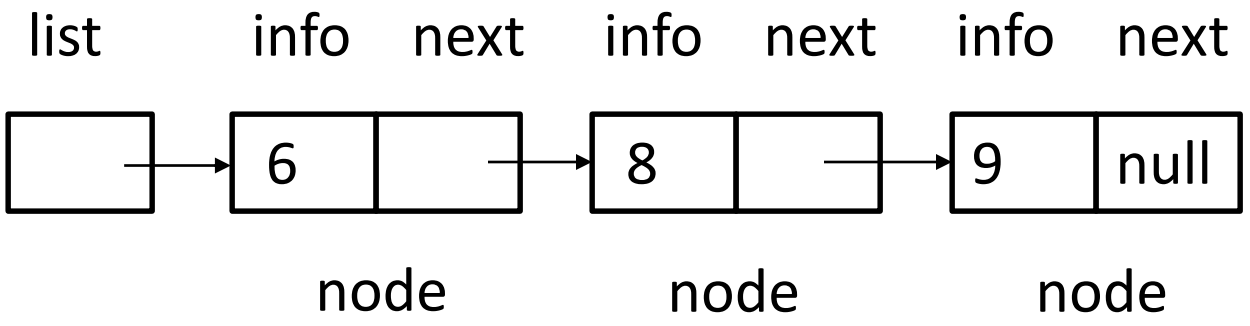
- Using sequential storage (such as array) to represent stacks and queues has following drawbacks:
  - Fixed amount of storage remains allocated to the stack or queue
  - No more than fixed amount of storage may be allocated, thus introducing the possibility of overflow
- A **linked list** is a dynamic data structure in which each item (called a **node**) within itself contains the address of the next item
- Each node contains two fields: an **information** field and a **next address** field

Linked Lists

- The information field holds the actual element on the list and the next address field contains the address of the next node in the list
- The address which is used to access a particular node is known as a **pointer**
- The entire linked list is accessed from an external pointer called **list** that points to (contains the address of) the first node in the list
- The next address field of the last node in the list contains a special value known as **null**, which is not a valid address and is used to signal the end of a list

Linked Lists

- The list with no nodes on it is called the empty list or the null list; The value of the external pointer to such list is the null pointer



Inserting and Removing Nodes

- A linked list is a dynamic data structure; We can insert and remove any number of nodes in a linked list
- The dynamic nature of a linked list may be contrasted with the static nature of an array, whose size remains constant
- To insert a node at a given point in the linked list, we create a new node with info (set to given information) and next (set to null) fields; Then we insert this new node at the given point
- To delete a node at a given point in a nonempty linked list, we set next field of its predecessor node (if any) to the address of its successor node

Inserting and Removing Nodes

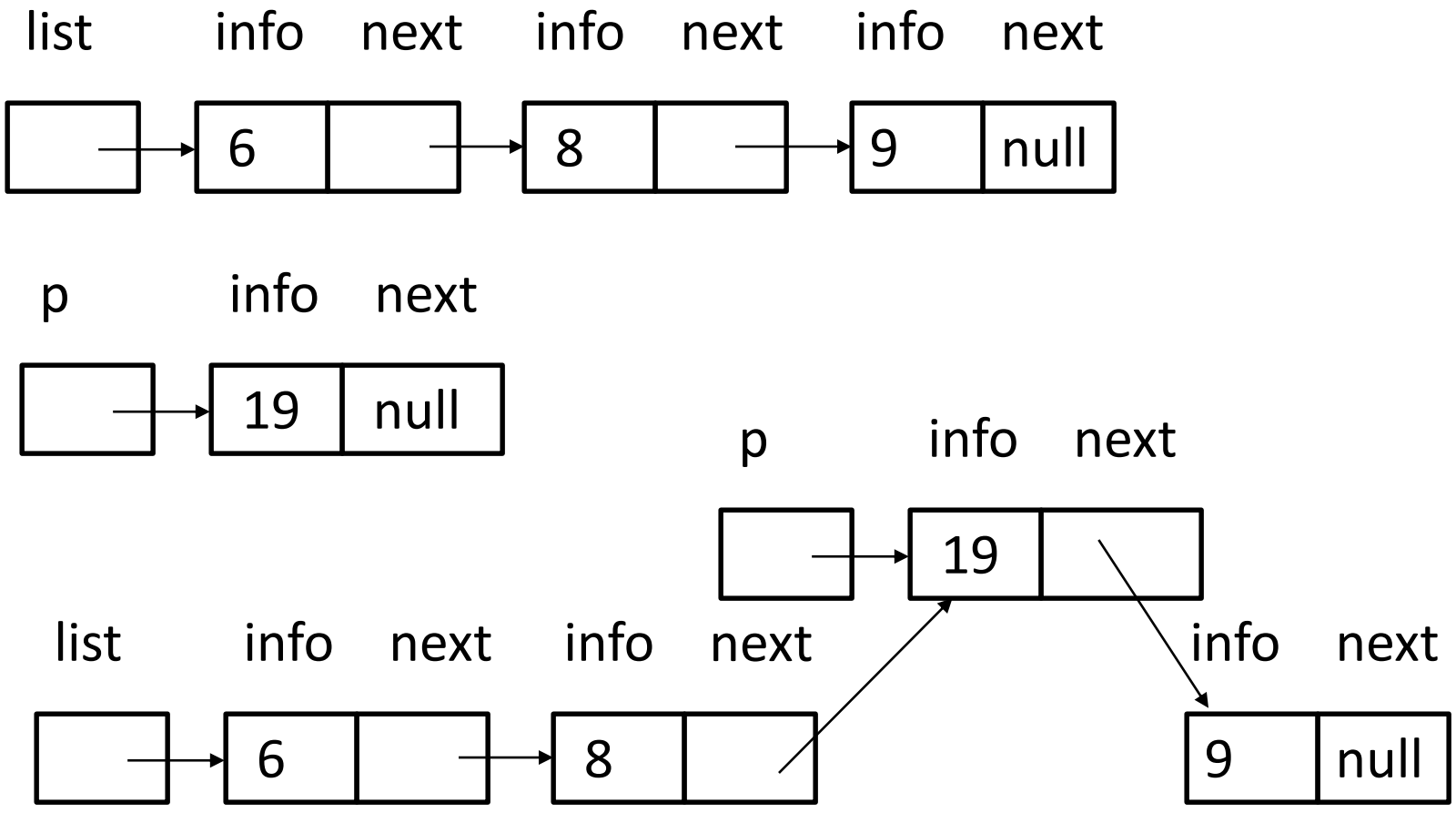


Fig: Inserting a node with information 19 after second node

Inserting and Removing Nodes

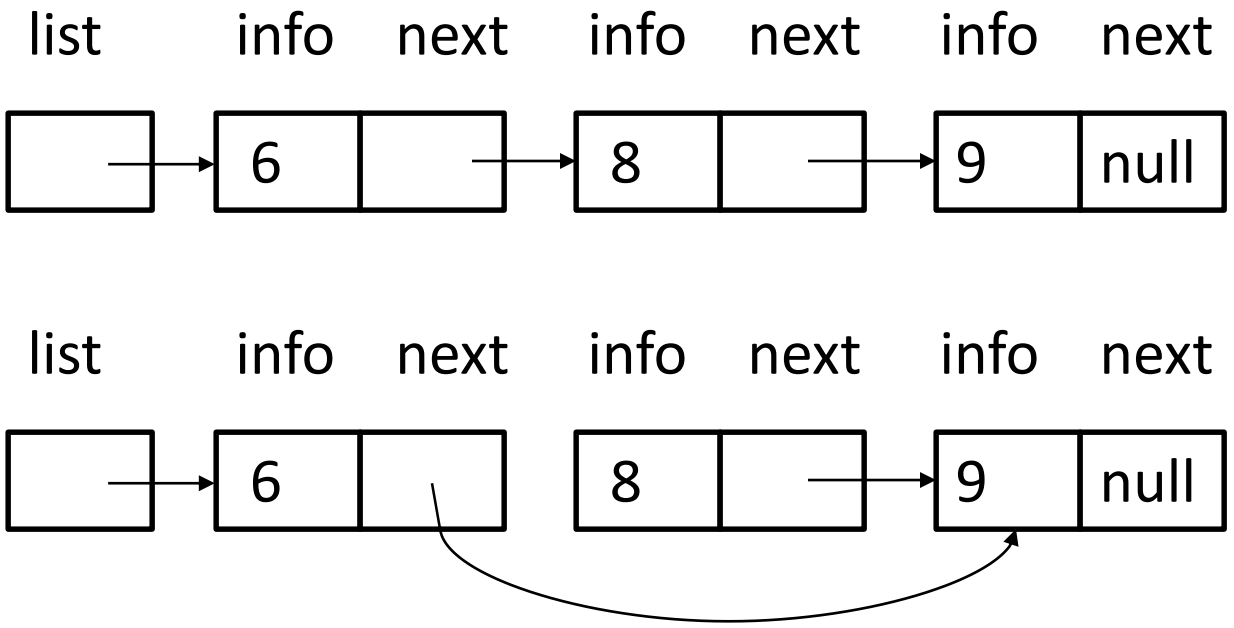


Fig: Deleting second node

C-Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node* next;
};
struct node* list = NULL;
void insertnode(struct node* pred, int val)
{
        struct node* ins = (struct node*) malloc(sizeof(struct node));
        ins->info = val;
        ins->next = NULL;
        if(pred == NULL)
```

C-Program

```
        {
                ins->next = list;
                list = ins;
        }
        else
        {
                ins->next = pred->next;
                pred->next = ins;
        }
}
void deletenode(struct node* del)
{
        if(del == list)
        {
                list  = del->next;
        }
        else
        {
                struct node* pred = list;
                while (pred->next != del)
                {
                        pred = pred->next;
                }
                pred->next = succ;
        }
        free(del);
}
```

C-Program

```
void display()
{
        struct node* temp = list;
        while(temp != NULL)
        {
                printf("%d\t", temp->info);
                temp = temp->next;
        }
}
```

```
int main()
{
        insertnode(NULL, 7);
        insertnode(list, 10);
        insertnode(list, 16);
        deletenode(list->next);
        display();
}
```