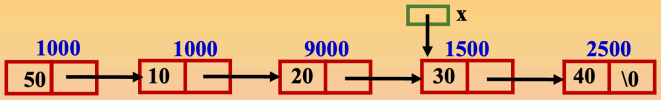
Why doubly linked lists are used?

Disadvantages of singly linked lists

- Using singly linked list and circular singly linked list, it is possible to traverse the list only in one direction. Hence, they are called one-way lists. It is not possible to traverse the list backwards i.e., two-way traversing is not possible.
- Insertion to the left of a designated node x is not possible. This is because, there is no way to find predecessor of x.



- In a singly linked list, deleting the specified node x is not possible unless we know the address of the first node. This is because, we do not know the address of predecessor of x. Given the address of the first node, we need to find the predecessor of x and then only we can delete the specified node x which is time consuming.

The above disadvantages can be overcome using doubly linked lists.

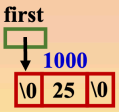
What is doubly linked list? Explain with example

Definition: A doubly singly linked list is a special type of linked list where traversing is done from left to right or right to left.

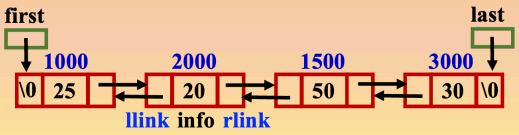
- Since traversing can be done from both the ends, it is also called two-way linked list.
- Each node has three fields namely:
  - info : contains the data to be manipulated.
  - llink : contains the address of left node.
  - rlink : contains the address of right node.
- The pictorial representation of doubly linked list is shown below:

Ex 1: Empty list

Ex 2: List with one node



Ex 3: List with more than one node

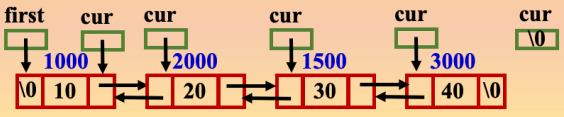


How to design a function to traverse a doubly linked list?

```
Algorithm display ( first )

Step 1:  //Check for empty list
         if ( first == NULL )
             print ("List is empty ")
             return
         end if

Step 2:  //Obtain the address of first node
         cur = first

Step 3:  //visit each node and print
         while ( cur != NULL )
             print ( info[cur] )
             cur = rlink[cur]
         end while

Step 4:  //Finished
         return
```

Case 1: List is empty

Case 2: List is existing



How to design a function to traverse a doubly linked list from first node?

```
Algorithm display ( first )

Step 1:  //Check for empty list
         if ( first == NULL )
             print ("List is empty ")
             return
         end if

Step 2:  //Obtain the address of first node
         cur = first

Step 3:  //visit each node and print
         while ( cur != NULL )
             print ( info[cur] )
             cur = rlink[cur]
         end while

Step 4:  //Finished
         return
```

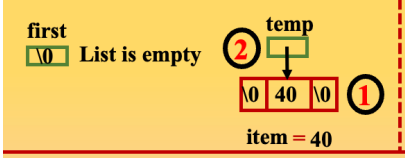
```
void display ( NODE first )
{
    NODE cur;
    printf ("List : " );
    //Check for empty list
    if ( first == NULL )
    {
        printf ("Empty ");
        return;
    }
    //Copy the address of first node
    cur = first;

    //visit each node and print
    while ( cur != NULL )
    {
        printf ("%d ---> ", cur ->info );
        cur = cur -> rlink;
    }
}
```

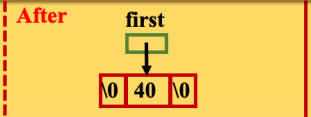

To display doubly linked list from the last node

```
void display ( NODE last )
{
    NODE cur;
    printf ("List : " );
    //Check for empty list
    if ( last == NULL )
    {
        printf ("Empty ");
        return;
    }
    //Copy the address of last node
    cur = last;

    //visit each node and print
    while ( cur != NULL )
    {
        printf ("%d ---> ", cur ->info );
        cur = cur -> llink;
    }
}
```

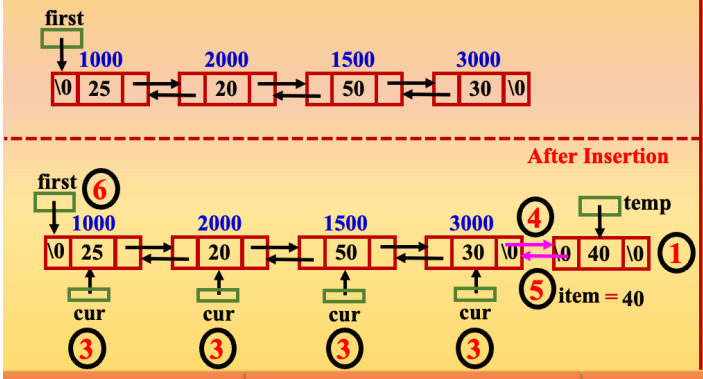
How to write a C function to insert an item at the rear end of the list?

Case 1: List is empty





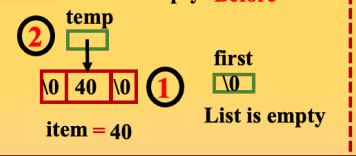
Case 2: List is non-empty

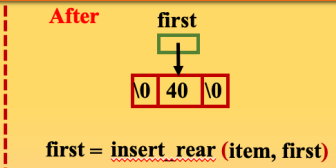



```
// Function to insert item at the rear end
NODE insert_rear ( int item, NODE first )
{
    NODE temp, cur ;
    // Create a node with item in it
    temp = getnode ( );
1   temp -> info = item;
    temp -> llink = temp -> rlink = NULL;
    // Insert the node for the first time
2   if ( first == NULL ) return temp;
    // Find the address of last node
    cur = first;
3   while ( cur -> rlink != NULL )
    {
        cur = cur -> rlink;
    }
    // Insert the node at the end of the list
4   cur -> rlink = temp ;
5   temp -> llink = cur ;
    // Return address of the first node
6   return first;
}
```

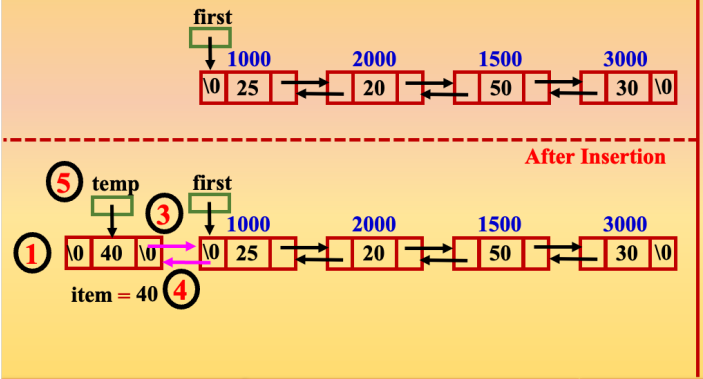

How to write a C function to insert an item at the front end of the list?

Case 1: List is empty





Case 2: List is non-empty

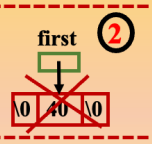



```
// Function to insert item at the rear end
NODE insert_front ( int item, NODE first )
{
    NODE temp;
    // Create a node with item in it
    temp = getnode ( );
1   temp -> info = item;
    temp -> llink = temp -> rlink = NULL;
    // Insert the node for the first time
2   if ( first == NULL ) return temp;
    // Insert the node at the front of the list
3   temp -> rlink = first ;
4   first -> llink = temp ;
    // Return address of the first node
5   return temp;
}
```

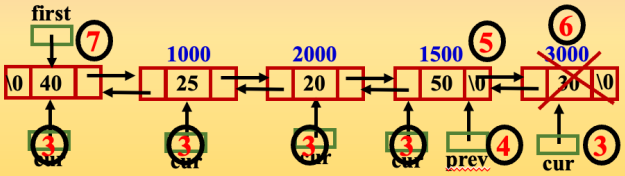
How to write a C function to delete an item from the rear end of the list?

Case 1: List is empty

Case 2: Only one node in the list



Case 3: More than one node in the list

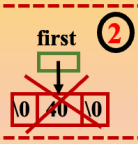



```
// Function to delete an item from the rear end of the list
NODE delete_rear (NODE first)
{
    NODE cur, prev;
(1) // Check for empty list
    if ( first == NULL )
    {
        printf ("List is empty\n");
        return NULL;
    }
(2) // Delete if there is only one node
    if ( first -> rlink == NULL )
    {
        printf (" Item deleted = %d \n ", first -> info);
        free ( first ), return NULL;
    }
(3) // Find address of last node
    cur = first;
    while ( cur -> rlink != NULL ) cur = cur -> rlink;
(4) prev = cur -> llink;        // Obtain last but one node
(5) // Make last but node as last node
    prev -> rlink = NULL;
(6) printf (" Item deleted = %d \n ", cur -> info);
    free ( cur );     // Delete the last node
(7) return first;
}
```

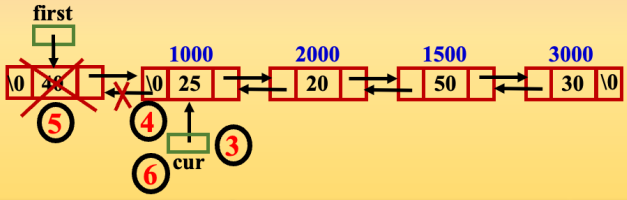

How to write a C function to delete an item from the front end of the list?

Case 1: List is empty

Case 2: Only one node in the list



Case 3: More than one node in the list




```
// Function to delete an item from the front end of the list
NODE delete_front (NODE first)
{
    NODE cur, prev;
(1) // Check for empty list
    if ( first == NULL )
    {
        printf ("List is empty\n");
        return NULL;
    }
(2) // Delete if there is only one node
    if ( first -> rlink == NULL )
    {
        printf (" Item deleted = %d \n ", first -> info);
        free ( first ), return NULL;
    }
(3) // Find address of second node
    cur = first -> rlink;
(4) // Make second node as the first node
    cur -> llink = NULL;
(5) // Delete the first node
    printf (" Item deleted = %d \n ", first -> info);
    free ( first );
(6) return cur;     // Return second node as first node
}
```

C program to implement double ended queue using doubly linked list?

```
#include <stdio..h>
#include <stdlib.h>

struct   node
{
        int                info;
        struct   node   * llink;
        struct   node   * rlink;
};

typedef   struct   node *   NODE;

NODE   getnode   ( ) ;

NODE   insert_rear   ( int   item ,   NODE   first ) ;
NODE   insert_front   ( int   item ,   NODE   first ) ;

NODE   delete_rear   (NODE   first ) ;
NODE   delete_front   (NODE   first ) ;

void      display ( NODE first ) ;
```

```
void   main  ( )
{
    int          choice,   item;
    NODE     first;

    first = NULL;
    for   ( ;; )
    {
        printf (  " 1:Insert Rear    2:Insert Front  3:Delete Rear : " );
        printf (  " 4:Delete Front   5:Display        6:Exit : "   );
        scanf (  " %d ",  &choice );

        switch (   choice )
        {
            case 1  :   printf (" Enter the item : " );
                        scanf ( " %d ",  &item );
                        first  =  insert_rear ( item,  first ) ;
                        break;
            case 2  :   printf ( " Enter the item : " );
                        scanf ( " %d ",  &item );
                        first  =  insert_front ( item,  first ) ;
                        break;
            case 3  :   first  =  delete_rear   ( first ) ;
                        break;
            case 4  :   last   =  delete_front   ( first ) ;
                        break;
            case 5  :   display ( first );
                        break;
            default:    exit ( 0 );
        }
    }
}
```


Why circular doubly linked lists are used?

Disadvantages of normal doubly linked lists

- To find the address of the last node, we need to traverse the list till the end. This is time consuming.
- Insertion and deletion at the rear end is time consuming.

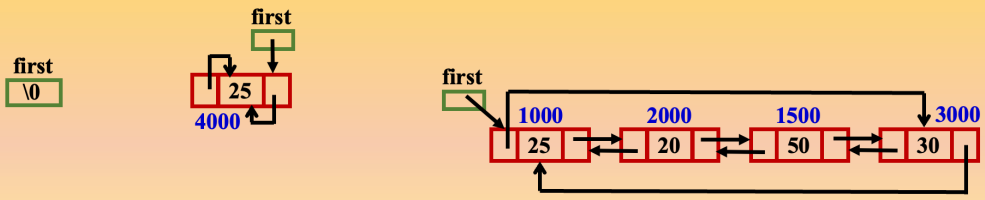
The above disadvantages can be overcome using circular doubly linked lists.

What is circular doubly linked list? Explain with example

Definition: A circular doubly singly linked list is a doubly linked list where the right link of last node contains address of the first node and left link of the first node contains address of the last node.

- The pictorial representation of circular doubly linked list is shown below:

Ex 1: Empty list **Ex 2:** List with one node **Ex 3:** List with more than one node

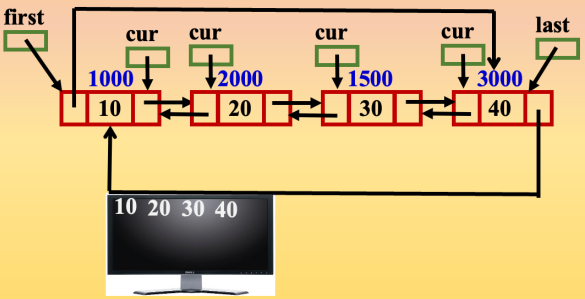


How to design a function to traverse a circular doubly linked list?

```
Algorithm display ( first )

Step 1:  //Check for empty list
         if ( first == NULL )
              print ("List is empty ")
              return
         end if

Step 2:  //Obtain the address of first node
         cur = first

Step 3:  //Obtain the address of last node
         last = llink[first]

Step 4:  //visit each node and print
         while ( cur != last )
              print ( info[cur] )
              cur = rlink [cur]
         end while
         print ( info[cur] )

Step 5:  //Finished
         return
```

Case 1: List is empty

Case 2: List is existing



How to design a function to traverse a circular doubly linked list?

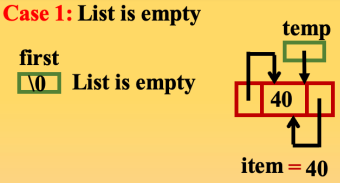
```
Algorithm display ( first )

Step 1:  //Check for empty list
         if ( first == NULL )
              print ("List is empty ")
              return
         end if

Step 2:  //Obtain the address of first node
         cur = first

Step 3:  //Obtain the address of last node
         last = llink[first]

Step 4:  //visit each node and print
         while ( cur != last )
              print ( info[cur] )
              cur = rlink [cur]
         end while
         print ( info[cur] )

Step 5:  //Finished
         return
```

```
void display ( NODE first )
{
      NODE cur, last;
      printf ("List : " );
      //Check for empty list
      if ( first == NULL )
      {
            printf ("Empty ");
            return;
      }
      //Obtain the address of first node
      cur = first;
      //Obtain the address of last node
      last = first–> llink;
      //visit each node and print
      while ( cur != last )
      {
            printf ("%d ---> ", cur –>info );
            cur = cur –> rlink;
      }
      printf ("%d \n", cur –>info );
}
```

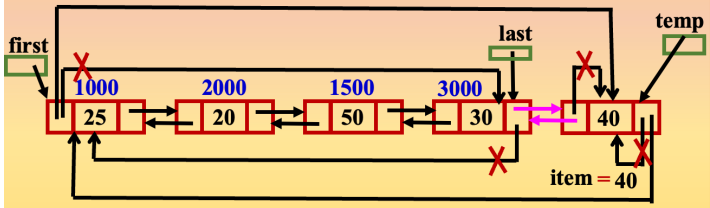
Dr. Padma Reddy A.M | Sai Vidya Institute of Technology | Bengaluru | download: nandipublications.com, saividya.ac.in

How to write a C function to insert an item at the rear end of the list?

Case 1: List is empty



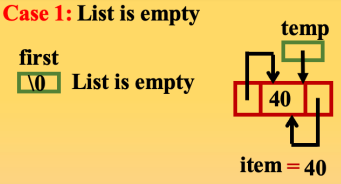
Case 2: List is non-empty



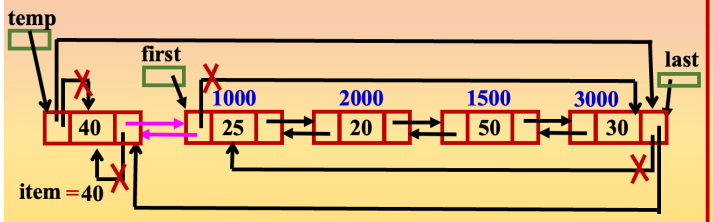

```
// Function to insert item at the rear end
NODE insert_rear ( int item, NODE first )
{
        NODE temp, last ;
        // Create a node with item in it
        temp = getnode ( );
        temp -> info = item;
        temp -> llink = temp -> rlink = temp;
        // Insert the node for the first time
        if ( first == NULL ) return temp;
        // Find the address of last node
        last = first->llink;
        // Insert the node at the end of the list
        last -> rlink = temp ;
        temp -> llink = last ;
        temp -> rlink = first ;
        first -> llink = temp ;
        // Return address of the first node
        return first;
}
```


How to write a C function to insert an item at the front end of the list?

Case 1: List is empty



Case 2: List is non-empty

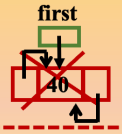



```
// Function to insert item at the front end
NODE insert_front ( int item, NODE first )
{
        NODE temp, last ;
        // Create a node with item in it
        temp = getnode ( );
        temp -> info = item;
        temp -> llink = temp -> rlink = temp;
        // Insert the node for the first time
        if ( first == NULL ) return temp;
        // Find the address of last node
        last = first->llink;
        // Insert the node at the front end of list
        temp-> rlink = first ;
        first -> llink = temp ;
        last -> rlink = temp ;
        temp -> llink = last ;
        // Return address of the first node
        return temp;
}
```

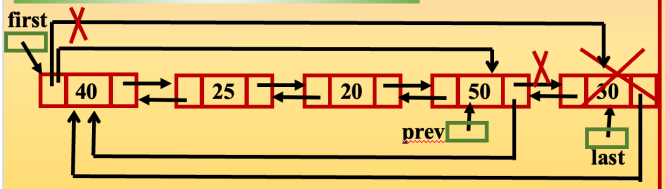
How to write a C function to delete an item from the rear end of the list?

Case 1: List is empty

Case 2: Only one node in the list



Case 3: More than one node in the list

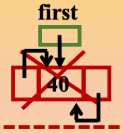



```
// Function to delete an item from the rear end of the list
NODE delete_rear (NODE first)
{
    NODE last, prev;

    // Check for empty list
    if (first == NULL)
    {
        printf ("List is empty\n");
        return NULL;
    }

    // Delete if there is only one node
    if (first -> rlink == first)
    {
        printf (" Item deleted = %d \n ", first -> info);
        free (first), return NULL;
    }

    last = first -> llink;      // get last node
    prev = last -> llink;       // Get last but one node

    // Make last but node as last node
    prev -> rlink = first;
    first -> llink = prev;
    printf (" Item deleted = %d \n ", last -> info);
    free (last);    // Delete the last node
    return first;
}
```

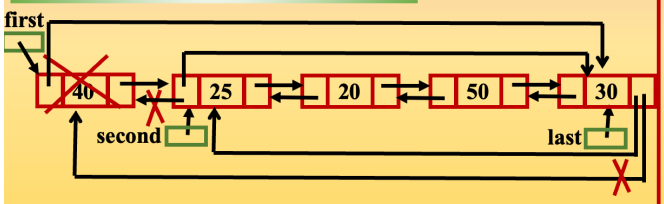

How to write a C function to delete an item from the front end of the list?

Case 1: List is empty

Case 2: Only one node in the list



Case 3: More than one node in the list




```
// Function to delete an item from the front end of the list
NODE delete_front (NODE first)
{
    NODE last, second;

    // Check for empty list
    if (first == NULL)
    {
        printf ("List is empty\n");
        return NULL;
    }

    // Delete if there is only one node
    if (first -> rlink == first)
    {
        printf (" Item deleted = %d \n ", first -> info);
        free (first), return NULL;
    }

    last = first -> llink;      // get last node
    second = first -> rlink;    // Get second node

    // Make second node as the first node
    last -> rlink = second;
    second -> llink = last;
    printf (" Item deleted = %d \n ", first -> info);
    free (first);   // Delete the first node
    return second;
}
```

C program to implement double ended queue using circular doubly linked list?

```
#include <stdio..h>
#include <stdlib.h>

struct   node
{
       int              info;
       struct   node   * llink;
       struct   node   * rlink;
};

typedef   struct   node *   NODE;

NODE   getnode   ( ) ;

NODE   insert_rear   ( int   item ,   NODE   first ) ;
NODE   insert_front   ( int   item ,   NODE   first ) ;

NODE   delete_rear   (NODE   first ) ;
NODE   delete_front   (NODE   first ) ;

void      display ( NODE first ) ;
```

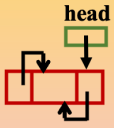
```
void   main  ( )
{
    int          choice,   item;
    NODE      first;

    first = NULL;
    for   ( ;; )
    {
        printf ( " 1:Insert Rear    2:Insert Front  3:Delete Rear : " );
        printf ( " 4:Delete Front   5:Display       6:Exit : "  );
        scanf ( " %d ",  &choice );

        switch (   choice )
        {
            case 1  :   printf ( " Enter the item : " );
                        scanf ( " %d ",  &item );
                        first  =  insert_rear ( item,  first ) ;
                        break;
            case 2  :   printf ( " Enter the item : " );
                        scanf ( " %d ",  &item );
                        first  =  insert_front ( item,  first ) ;
                        break;
            case 3  :   first  =  delete_rear  ( first ) ;
                        break;
            case 4  :   first  =  delete_front  ( first ) ;
                        break;
            case 5  :   display ( first );
                        break;
            default:    exit ( 0 );
        }
    }
}
```


What is circular doubly linked list with header? Explain with example

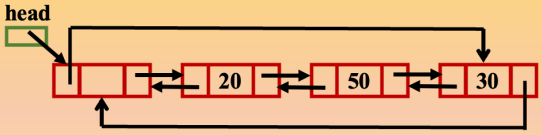
Definition: A circular doubly singly linked list with header is a circular doubly linked list where the right link of last node contains address of the header node and left link of the header node contains address of the last node.

❖ The pictorial representation of circular doubly linked list is shown below:

Ex 1: Empty list



Ex 2: Non empty list



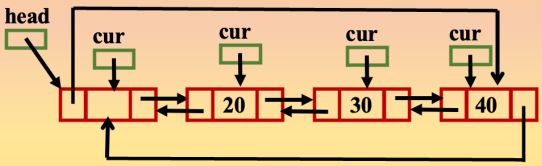
How to design a function to traverse a circular doubly linked list with header node?

Algorithm display (head)

```
Step 1:   //Check for empty list
          if ( rlink[head]==  head )
               print ("List is empty ")
               return
          end if

Step 2:   //Obtain the address of first node
          cur  =  rlink[head]

Step 3:   //visit each node and print
          while (cur   !=   head)
               print ( info[cur] )
               cur  =  rlink[cur]
          end while

Step 5:   //Finished
          return
```

Case 1: List is empty

Case 2: List is existing



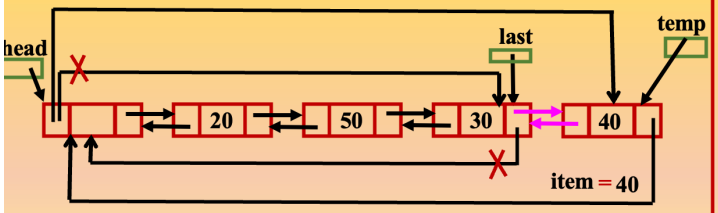
How to design a function to traverse a circular doubly linked list with header node?

```
Algorithm display ( head )

Step 1:  //Check for empty list
         if ( rlink[head]== head )
              print ("List is empty ")
              return
         end if

Step 2:  //Obtain the address of first node
         cur = rlink[head]

Step 3:  //visit each node and print
         while (cur != head)
              print ( info[cur] )
              cur = rlink[cur]
         end while

Step 5:  //Finished
         return
```

```
void display ( NODE head )
{
      NODE cur;
      printf ("List : " );
      //Check for empty list
      if ( head->rlink== head )
      {
            printf ("Empty ");
            return;
      }
      //Obtain the address of first node
      cur = head->rlink;
      //visit each node and print
      while ( cur != head )
      {
            printf ("%d ---> ", cur->info );
            cur = cur -> rlink;
      }
}
```

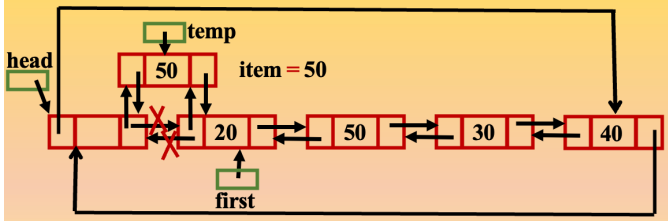
Dr. Padma Reddy A.M | Sai Vidya Institute of Technology | Bengaluru | download: nandipublications.com, saividya.ac.in

How to write a C function to insert an item at the rear end of the list?




```
// Function to insert item at the rear end
NODE insert_rear ( int item, NODE head)
{
      NODE temp, last ;
      // Create a node with item in it
      temp = getnode ( );
      temp -> info = item;

      // Find the address of last node
      last = head->llink;

      // Insert the node at the end of the list
      last -> rlink = temp ;
      temp -> llink = last ;
      temp -> rlink = head ;
      head -> llink = temp ;

      // Return address of the head node
      return head;
}
```

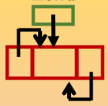
How to write a C function to insert an item at the front end of the list?



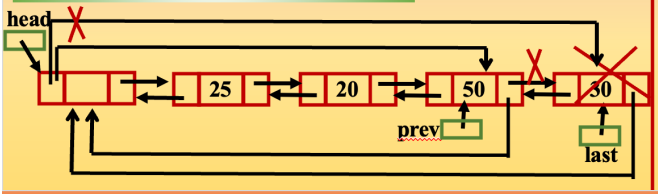

```
// Function to insert item at the front end
NODE insert_front ( int item, NODE head)
{
      NODE temp, first ;
      // Create a node with item in it
      temp = getnode ( );
      temp -> info = item;

      // Find the address of first node
      first = head->rlink;

      // Insert the node at the front end of list
      temp->rlink = first ;
      first -> llink = temp ;
      head -> rlink = temp ;
      temp -> llink = head ;

      // Return address of the header node
      return head;
}
```

How to write a C function to delete an item from the rear end of the list?

Case 1: List is empty



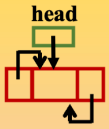
Case 2: List is existing



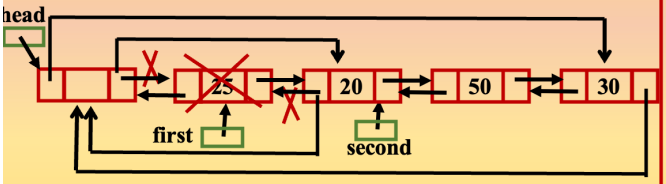

```
// Function to delete an item from the rear end of the list
NODE delete_rear (NODE head)
{
        NODE  last, prev ;
        // Check for empty list
        if ( head->rlink == head )
        {
             printf (" List is empty " );
             return  head;
        }
        last =  head-> llink;       // get last node
        prev =  last -> llink;      // Get last but one node
        // Make last but node as last node
        prev -> rlink  = head ;
        head-> llink  = prev ;
        printf (" Item deleted =  %d  \n ", last -> info);
        free  ( last );     // Delete the last node
        return  head;
}
```

How to write a C function to delete an item from the front end of the list?

Case 1: List is empty



Case 3: List is existing




```
// Function to delete an item from the front end of the list
NODE delete_front ( NODE head )
{
        NODE  first, second ;
        // Check for empty list
        if ( head ->rlink == head )
        {
             printf (" List is empty");
             return  head;
        }
        first =  head ->rlink;      // get first node
        second = first  -> rlink;   // Get second node
        // Make second node as the first node
        head ->rlink  = second ;
        second-> llink  = head ;
        printf (" Item deleted =  %d  \n ", first-> info);
        free  ( first );    // Delete the first node
        return  head;
}
```

Dr. Padma Reddy A.M | Sai Vidya Institute of Technology | Bengaluru | download: nandipublications.com, saividya.ac.in

What is a polynomial? What is the degree of a polynomial?

Definition: A polynomial is sum of terms where each term has a form :

$$a\,x^n$$

where

- a : is a coefficient
- x : is a variable
- n : is exponent value

■ The largest exponent value in the polynomial is called leading exponent. It is also called degree of a polynomial.

Ex 1: $A(x) = 6x^{25} + 5x^{10} + 35$
- ► It is sum of three terms.
- ► 25 is the largest exponent in this polynomial. So, degree of polynomial is 25

Ex 2: $B(x) = 9x^4 - 5x^3 + 6x^2 - 20$
- ► It is sum of four terms.
- ► 4 is the largest exponent in this polynomial. So, degree of polynomial is 4

How to represent a polynomial using array of structures?

- Polynomial: $A(x) = 9x^4 - 5x^3 + 6x^2 - 20$
- Each term has two fields:
  - Coefficient field: c
  - power of x field: px

The structure definition for a term can be written as:

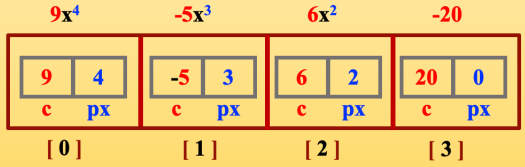
```
typedef  struct
{
    int   c;
    int   px;
} POLY ;
```

- A polynomial has one or more terms or it is a collection of one or more terms. So, the data structure that we can think of is an array. The above polynomial can be initialized using array of structures as shown below:

$9x^4$ $-5x^3$ $6x^2$ -20

```
POLY  a[ ] = { { 9, 4 },  { -5, 3 },  { 6, 2 },  { -20, 0 } }
```

- The pictorial representation of given polynomial can be written as shown below:



- ith term can be accessed by writing: a[i]
- coefficient of ith can be accessed by writing: a[i].c
- power of x of ith can be accessed by writing: a[i].px

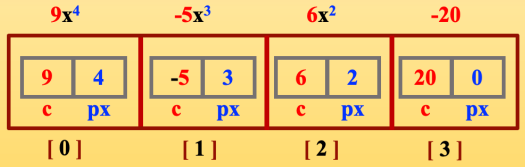
How to display a polynomial?

```
void  print_polynomial  ( POLY   a[],  int   n )
{
    int   i;
    for ( i = 0;  i < n;  i++)
    {
       if ( a[i].c  > 0 )   +9 x ^ 4
            printf ( " + %d x ^ %d ",  a[i].c , a[i].px);
       else
            printf (  "  %d x ^ %d ",  a[i].c , a[i].px);
    }
    printf ("\n ");
}
```

$9x^4$ $-5x^3$ $6x^2$ -20

```
POLY  a[ ] = { { 9, 4 },  { -5, 3 },  { 6, 2 },  { -20, 0 } }
```

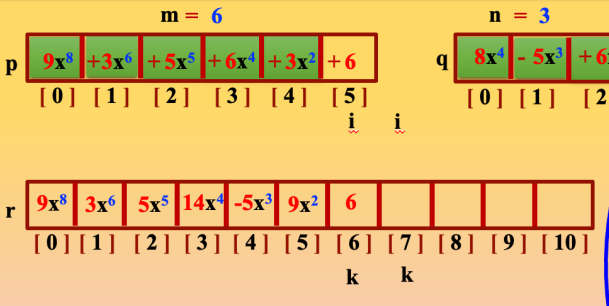
- The pictorial representation of given polynomial can be written as shown below:



- ith term can be accessed by writing: a[i]
- coefficient of ith can be accessed by writing: a[i].c
- power of x of ith can be accessed by writing: a[i].px


```
void  read_polynomial  ( POLY   a[],  int   n )
{
      int   i;
      for ( i = 0;  i < n;  i++)
      {
            scanf ("%d  %d ", &a[i].c,  &a[i].px);
      }
}
```

How to design a function to add two polynomials?




```
Algorithm add_2_polynomials ( p, m, q, n, r )
i = j = k = 0
while ( i < m && j < n )
    Case 1: if ( p[i].px == q[j].px )
                sum = p[i].c + q[j].c
                if ( sum != 0 )
                    r[k].c = sum , r[k].px = p[i].px
                    k++
                i++, j++
    Case 2: if ( p[i].px > q[j].px )
                r[k].c = p[i].c , r[k].px = p[i].px
                i++, k++
    Case 3: r[k].c = q[j].c , r[k].px = q[j].px
                j++, k++
r[k].c = p[i].c , r[k].px = p[i].px          while ( i < m )
i++, k++
r[k].c = q[j].c , r[k].px = q[j].px          while ( j < n )
i++, k++
return k
```


How to write a C function to add two polynomials?

```
Algorithm add_2_polynomials ( p, m, q, n, r )
i = j = k = 0
while ( i < m && j < n )
    Case 1: if ( p[i].px == q[j].px )
                sum = p[i].c + q[j].c
                if ( sum != 0 )
                    r[k].c = sum , r[k].px = p[i].px
                    k++
                i++, j++
    Case 2: if ( p[i].px > q[j].px )
                r[k].c = p[i].c , r[k].px = p[i].px
                i++, k++
    Case 3: r[k].c = q[j].c , r[k].px = q[j].px
                j++, k++
r[k].c = p[i].c , r[k].px = p[i].px          while ( i < m )
i++, k++
r[k].c = q[j].c , r[k].px = q[j].px          while ( j < n )
i++, k++
return k
```

```
int add_2_polynomials (POLY p [], int m,
{                       POLY [] b , int n , POLY c [] )
    int i = 0, j = 0, k = 0, sum;
    while ( i < m && j < n ) {
        if ( p[i].px == q[j].px ) {
            sum = p[i].c + q[j].c ;
            if ( sum != 0 ) {
                r[k].c = sum, r[k].px = p[i].px ;
                k++ ;
            }
            i++, j++;
        } else if ( p[i] .px > q[j] .px ) {
            r[k].c = p[i].c , r[k].px = p[i].px ;
            i++, k++ ;
        } else {
            r[k].c = q[j].c , r[k] .px = q[j] .px ;
            j++, k++ ;
        }
    }
    while ( i < m ) {
        r[k].c = p[i].c, r[k].px = p[i].px ;
        i++ , k++ ;
    }
    while ( j < n) {
        r[k].c = q[j].c , r[k] .px = q[j] .px;
        j++, k++;
    }
    return k;
}
```

How to write a C function to add two polynomials?

```c
#include <stdio.h>
typedef struct
{
    int c;
    int px;
} POLY ;
void print_polynomial ( POLY a[], int n )
{
    int i;
    for ( i = 0; i < n; i++ )
    {
        if ( a[i].c > 0 )
            printf ( " + %d x ^ %d ", a[i].c, a[i].px);
        else
            printf ( "  %d x ^ %d ", a[i].c, a[i].px);
    }
    printf ("\n");
}
void read_polynomial ( POLY a[], int n )
{
    int i;
    for ( i = 0; i < n; i++ )
    {
        scanf ("%d  %d ", &a[i].c,  &a[i].px);
    }
}
// Insert the function to add two polynomials
int add_2_polynomials (POLY p[], int m, POLY [] b, int n, POLY c [] )
```

```c
int add_2_polynomials (POLY p[], int m,
                       POLY [] b, int n, POLY c [] )
{
    int i = 0, j = 0, k = 0, sum;
    while ( i < m && j < n ) {
        if ( p[i].px == q[j].px ) {
            sum = p[i].c + q[j].c;
            if ( sum != 0 ) {
                r[k].c = sum, r[k].px = p[i].px;
                k++ ;
            }
            i++, j++;
        } else if ( p[i].px > q[j].px ) {
            r[k].c = p[i].c , r[k].px = p[i].px;
            i++, k++;
        } else {
            r[k].c = q[j].c , r[k].px = q[j].px ;
            j++, k++;
        }
    }
    while ( i < m ) {
        r[k].c = p[i].c, r[k].px = p[i].px ;
        i++ , k++ ;
    }
    while ( j < n) {
        r[k].c = q[j].c , r[k].px = q[j].px;
        j++, k++;
    }
    return k;
}
```


How to write a C function to add two polynomials?

```c
#include <stdio.h>
typedef struct
{
    int c;
    int px;
} POLY ;
void print_polynomial ( POLY a[], int n )
{
    int i;
    for ( i = 0; i < n; i++ )
    {
        if ( a[i].c > 0 )
            printf ( " + %d x ^ %d ", a[i].c, a[i].px);
        else
            printf ( "  %d x ^ %d ", a[i].c, a[i].px);
    }
    printf ("\n");
}
void read_polynomial ( POLY a[], int n )
{
    int i;
    for ( i = 0; i < n; i++ )
    {
        scanf ("%d  %d ", &a[i].c,  &a[i].px);
    }
}
// Insert the function to add two polynomials
int add_2_polynomials (POLY p[], int m, POLY [] b, int n, POLY c [] )
```

```c
void main ( )
{
    POLY a[20], b[20], c[40];
    int  m, n, x;

    printf ( " Enter the no. of terms in 1st polynomial : " );
    scanf ( "%d ", &m );

    printf ( " Enter the terms of 1st polynomial : ");
    read_polynomial ( a, m );

    printf ( " Enter the no. of terms in 2nd polynomial : " );
    scanf ( "%d ", &n );

    printf ( " Enter the terms of 2nd polynomial : ");
    read_polynomial ( b, n );

    x = add_2_polynomials (a, m, b, n, c);

    printf ( " Polynomial 1 : " );
    print_polynomial ( a, m );

    printf ( " Polynomial 2 : " );
    print_polynomial ( b, n );

    printf ( " Polynomial 3 : " );
    print_polynomial ( c, x );
}
```

Dr. Padma Reddy A.M | Sai Vidya Institute of Technology | Bengaluru | download: nandipublications.com, saividya.ac.in

How to write a C function to add two polynomials?

Test case: 1

a (x) = $9x^8 + 3x^6 + 5x^5 + 6x^4 + 3x^2 + 6$ m = 6

b (x) = $8x^4 - 5x^3 + 6x^2$ n = 3

c (x) = $9x^8 + 3x^6 + 5x^5 + 14x^4 - 5x^3 + 9x^2 + 6$ k = 7

Run 1

Enter the no of terms in 1st polynomial : 6

Enter the terms of 1st polynomial: 9 8 3 6 5 5 6 4 3 2 6 0

Enter the no of terms in 2nd polynomial : 3

Enter the terms of 2nd polynomial: 8 4 -5 3 6 2

Polynomial 1: + 9x^8 + 3x^6 + 5x^5 + 6x^4 + 3x^2 + 6x^0

Polynomial 2: + 8x^4 - 5x^3 + 6x^2

Polynomial 3: + 9x^8 + 3x^6 + 5x^5 + 14x^4 - 5x^3 + 9x^2 + 6x^0

```
void main ( )
{
    POLY a[20], b[20], c[40];
    int  m, n, x;

    printf ( " Enter the no. of terms in 1st polynomial : " );
    scanf ( "%d ", &m );

    printf ( " Enter the terms of 1st polynomial : ");
    read_polynomial ( a, m );

    printf ( " Enter the no. of terms in 2nd polynomial : ");
    scanf ( "%d ", &n );

    printf ( " Enter the terms of 2nd polynomial : ");
    read_polynomial ( b, n );

    x = add_2_polynomials (a, m, b, n, c);

    printf ( " Polynomial 1 : " );
    print_polynomial ( a, m );

    printf ( " Polynomial 2 : " );
    print_polynomial ( b, n );

    printf ( " Polynomial 3 : " );
    print_polynomial ( c, x );
}
```

How to write a C function to add two polynomials?

Test case: 2

a (x) = $3x^4 + 3x^3 - 5x^2$ m = 3

b (x) = $8x^4 - 3x^3 + 5x^2 + 3x + 6$ n = 5

c (x) = $11x^4 + 3x + 6$ x = 3

Run 1

Enter the no of terms in 1st polynomial : 3

Enter the terms of 1st polynomial: 3 4 3 3 -5 2

Enter the no of terms in 2nd polynomial : 5

Enter the terms of 2nd polynomial: 8 4 -3 3 5 2 3 1 6 0

Polynomial 1: + 3x^4 + 3x^3 - 5x^2

Polynomial 2: + 8x^4 - 3x^3 + 5x^2 + 3x^1 + 6x^0

Polynomial 3: + 11x^4 + 3x^1 + 6x^0

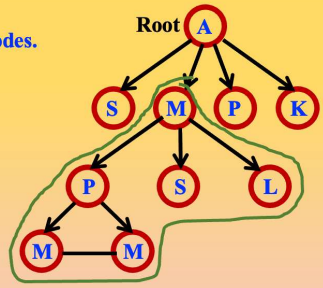
```
void main ( )
{
    POLY a[20], b[20], c[40];
    int  m, n, x;

    printf ( " Enter the no. of terms in 1st polynomial : " );
    scanf ( "%d ", &m );

    printf ( " Enter the terms of 1st polynomial : ");
    read_polynomial ( a, m );

    printf ( " Enter the no. of terms in 2nd polynomial : ");
    scanf ( "%d ", &n );

    printf ( " Enter the terms of 2nd polynomial : ");
    read_polynomial ( b, n );

    x = add_2_polynomials (a, m, b, n, c);

    printf ( " Polynomial 1 : " );
    print_polynomial ( a, m );

    printf ( " Polynomial 2 : " );
    print_polynomial ( b, n );

    printf ( " Polynomial 3 : " );
    print_polynomial ( c, x );
}
```

What is a tree?

❖ Tree is a non-linear data structure where nodes are linked to each other in parent-child relationship such that there is only one path between any given two nodes.
- There is a special node called root node for which there is no parent.
- Remaining nodes are partitioned into subtrees

❖ Tree is also defined as acyclic directed graph

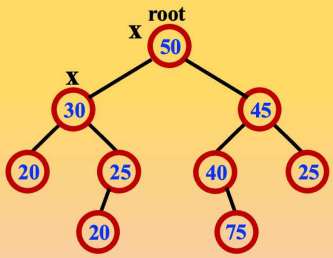
Ex : In the tree shown in figure on right hand side:
- The tree has 10 nodes: A, S, M, P, K, P, S, L, M, M
- Node A is root node and it is written at the top.
- Nodes S, M, P, K are children of node A and hence there are four subtrees identified by S, M, P, K



Note: An m-ary tree is defined as a tree where each node has maximum of m children.
- If m = 2, the tree is 2-ary tree or bin-ary tree.
- If m = 3, the tree is 3-ary tree or tern-ary tree.
- If m = 4, the tree is 4-ary tree or quad-ary tree.

What are basic tree terminologies?

Root node: A node in a tree which has no parent is called root node.
- ❖ Root node is the topmost node in a tree.
- ❖ Using root node, any node in the tree can be accessed.
- ❖ There is only one root node in a tree.
- ❖ In the given tree node containing item 50 is the root node.



Descendants: The nodes that are reachable from node x while moving downwards are called descendants of node x.
- ❖ All the nodes below 30 i.e, 20, 25 and 20 are descendants of 30.
- ❖ All the nodes below 45 i.e, 40, 25 and 75 are descendants of 45.

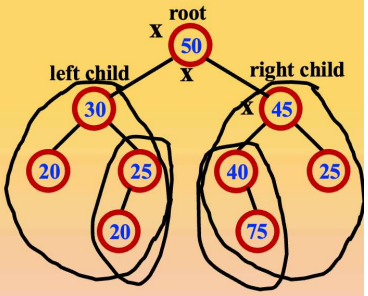
Left descendants: The nodes that are reachable from left side of node x while moving downwards are called left descendants of node x.
- ❖ The nodes 30, 20, 25 and 20 are left descendants of 50.
- ❖ The nodes 40 and 75 are left descendants of 45.

Right descendants: The nodes that are reachable from right side of node x while moving downwards are called right descendants of node x.
- ❖ The nodes 45, 40, 25 and 75 are right descendants of 50.
- ❖ The nodes 25 and 20 are right descendants of 30.

What are basic tree terminologies?

Left subtree: All the nodes that are all left descendants of node x form the left subtree of x.
- ❖ The nodes 30, 20, 25 and 20 together form the left subtree of node 50.
- ❖ The nodes 40 and 75 together form the left subtree of node 45.

Right subtree: All the nodes that are all right descendants of node x form the right subtree of x.
- ❖ The nodes 45, 40, 25 and 75 together form the right subtree of node 50.
- ❖ The nodes 25 and 20 together form the right subtree of node 30.



Child: A node which is the first descendant of a given node x is the child of node x.
- ❖ A node which is the first left descendant of a node is called left child.
- ❖ A node which is the first right descendant of a node is called right child.

What are basic tree terminologies?

Left subtree: All the nodes that are all left descendants of node x form the left subtree of x.

- The nodes 30, 20, 25 and 20 together form the left subtree of node 50.
- The nodes 40 and 75 together form the left subtree of node 45.

Right subtree: All the nodes that are all right descendants of node x form the right subtree of x.

- The nodes 45, 40, 25 and 75 together form the right subtree of node 50.
- The nodes 25 and 20 together form the right subtree of node 30.

Child: A node which is the first descendant of a given node x is the child of node x.

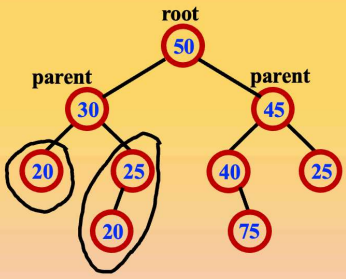
- A node which is the first left descendant of a node is called left child.
- A node which is the first right descendant of a node is called right child.

Parent: A node having left subtree or right subtree or both is said to be a parent.

- The node 30 is the parent of nodes 20 and 25
- The node 45 is the parent of nodes 40 and 25

Siblings: The nodes having the same parent are called siblings.

- The nodes 20 and 25 are siblings.
- The nodes 40 and 25 are siblings.



What are basic tree terminologies?

Ancestors: The nodes that are reachable from node x while moving upwards are called ancestors of node x.

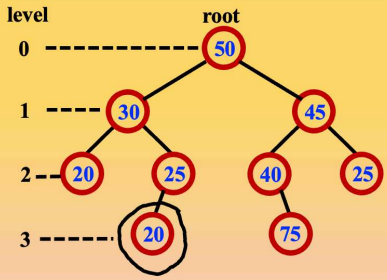
- All the nodes above 20 i.e. 25, 30 and 50 are ancestors of 20.
- All the nodes above 75 i.e. 40, 45 and 50 are ancestors of 75.

Leaf / external node: A node having empty left child and empty right child is called a leaf node or a terminal node or an external node.

- The nodes 20, 20, 75 and 25 are all leaf nodes.

Internal nodes: The nodes except leaf nodes are called internal nodes.

- The nodes 25, 40, 30, 45 and 50 are all internal nodes.



Level: The total number of edges from root to a node is called level of a node or depth of a node

- The total number of edges from 75 to root = 3. So, level of 75 is 3.

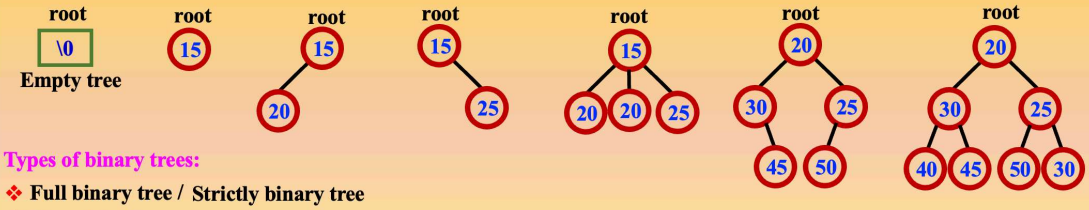
Height: The total number of nodes from a farthest leaf node to root is called height of a tree or depth of a tree.

- The total number of nodes from 75 to root = 4. So, height of tree is 4.
- The height of the tree = Number of levels = 4.

What is a binary tree? What are the different types of binary trees?

Definition: An m-ary tree where m = 2 is called 2-ary tree or bin-ary tree or binary tree.

- In other words, a tree where each node in the tree has maximum of two children is called binary tree.
- Each node in a binary tree can have either 0, 1 or 2 children but, a node can not have more than two children.
- An empty tree can also be considered as binary tree.
- For example, the binary trees are shown below:



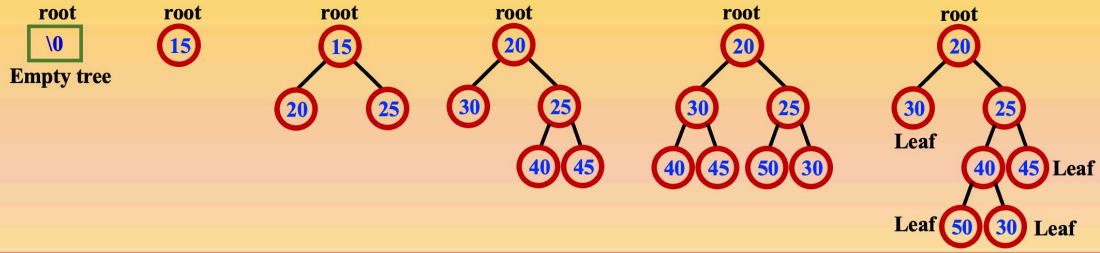
Types of binary trees:

- Full binary tree / Strictly binary tree
- Complete binary tree
- Almost complete binary tree
- Binary search tree
- AVL trees
- B-trees
- RED-BLACK-trees

What is a Full binary tree?

Definition: A binary tree where each node has either 0 or 2 children is called full binary tree or strictly binary tree.
In other words, a binary tree in which all the nodes have two children except the leaf nodes is called full binary tree.

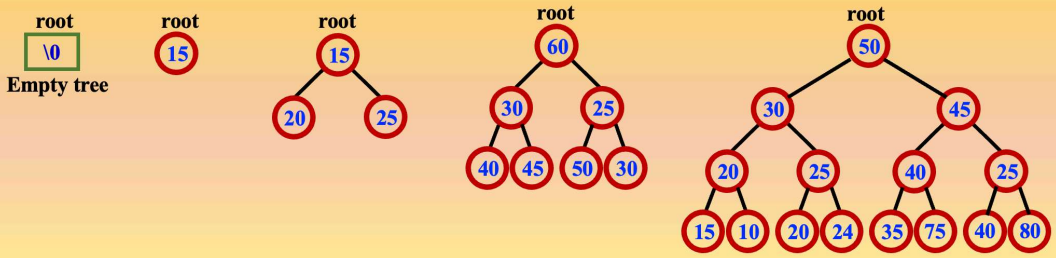
- An empty tree can also be considered as full binary tree.
- For example, all following binary trees are full binary trees:



What is a complete binary tree?

Definition: A binary tree where each node has either 0 or 2 children (full binary tree or strictly binary tree) and all the leaf nodes are at the same level is called complete binary tree.

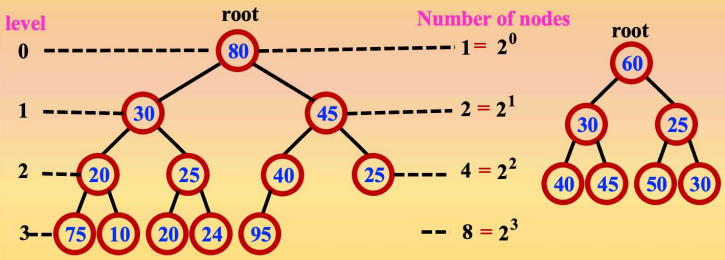
- An empty tree is considered as complete binary tree.
- At any level i in a complete binary tree the number of nodes = 2^i
- For example, all following binary trees are complete binary trees:



What is an almost complete binary tree?

Definition: A binary tree is an almost complete binary tree with the following properties:

- If i is the level of the tree, the number of nodes in i^{th} level must be 2^i
- If number of nodes in i^{th} level $< 2^i$ then the number of nodes in $(i-1)^{th}$ level must be 2^{i-1} and all the nodes i^{th} level must be filled from left to right only.
- A node in an almost complete binary tree cannot have right child without having left child. But, a node can have only left child.



How to represent a tree?

A tree can be represented using three different ways:

- List representation
- Left-child Right sibling representation
- Left child – Right child (Degree 2) representation

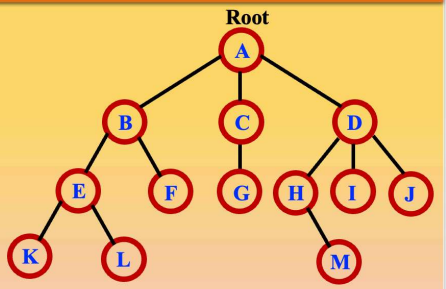
What is list representation of a tree?

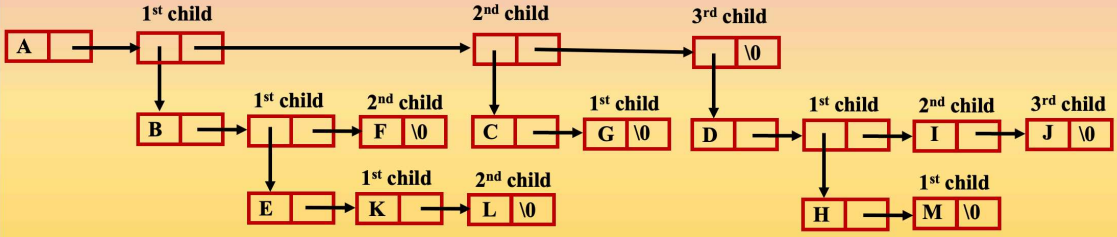
A tree can be represented using list as shown below:

- The root node comes first.
- It is immediately followed by a list of subtrees of that node
- It is recursively repeated for each subtree.

Observe the following points:

- There are 3 children for node A in the tree. So, there are 3 nodes to the right of A in list representation.
- A's first child is B, 2nd child is C and 3rd child is D and they are shown using down links.

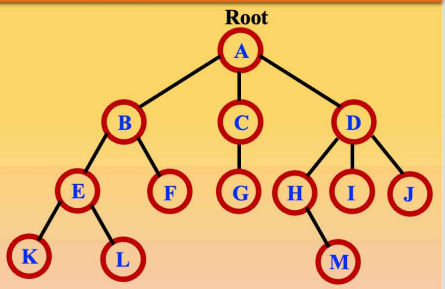


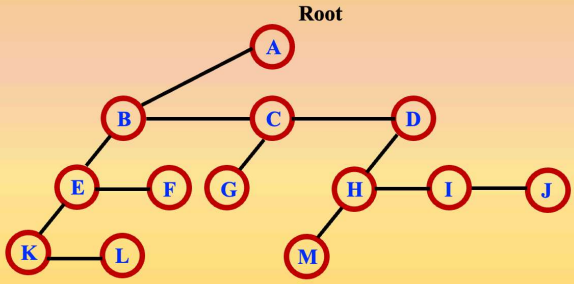


How to represent a tree using left-child right sibling representation?

A left child right sibling representation of a tree can be obtained as shown below:

- The root node comes first.
- The left pointer of a node in the tree will be the left child in this representation
- The remaining children of a node in the tree (siblings) can be inserted horizontally to the left child in the representation.





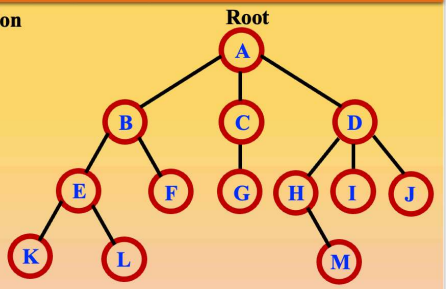
Observe the following points:

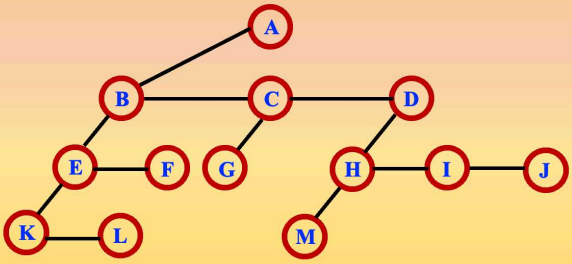
- A's left child is B in the tree. So, A's left child is B in the representation.
- A's remaining children such as C and D in the tree are inserted horizontally to node B in the representation.

How to represent a tree in Left child – right child (degree 2) representation?

A tree can be represented as left child – right child or degree 2 representation as shown below:

- Obtain the left-child right sibling representation.
- Rotate the horizontal lines clockwise by 45 degrees.

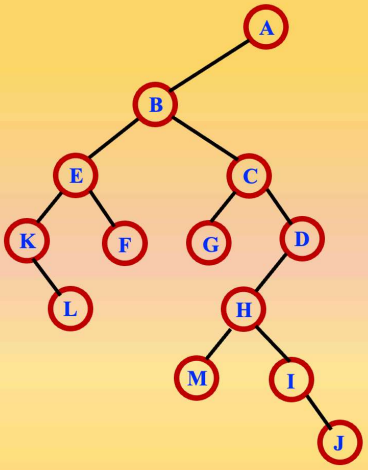


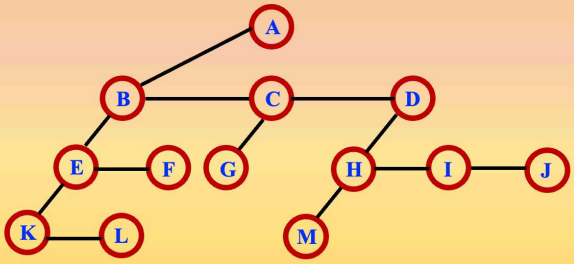


How to represent a tree in Left child – right child (degree 2) representation?

A tree can be represented as left child – right child or degree 2 representation as shown below:

- Obtain the left-child right sibling representation.
- Rotate the horizontal lines clockwise by 45 degrees.

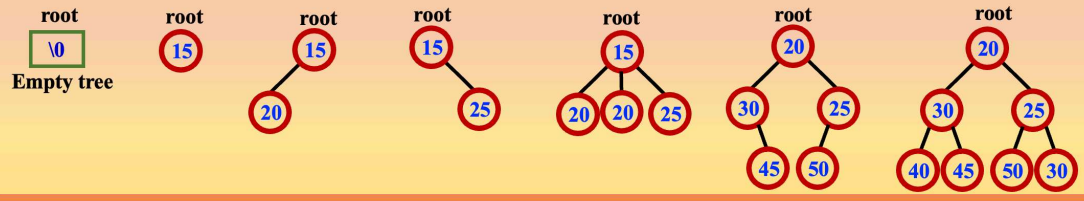




What is a binary tree?

Definition: An m-ary tree where m = 2 is called 2-ary tree or bin-ary tree or binary tree.

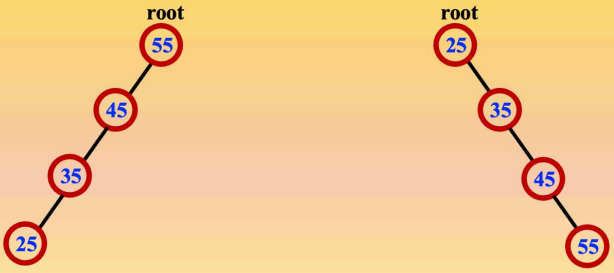
- In other words, a tree which has finite set of nodes that is either empty or consists of a root node and each node in the tree has maximum of two children i.e., left subtree and right subtree is called binary tree.
  - Root : A node without a parent is called root node. It is the first node in the tree.
  - Left subtree : A tree connected to left side of a node is called left subtree.
  - Right subtree : A tree connected to right side of a node is called right subtree.
- Each node in a binary tree can have either 0, 1 or 2 children but, a node can not have more than two children.
- An empty tree can also be considered as binary tree.
- For example, the binary trees are shown below:



What is skewed binary tree?

Definition: A skewed binary tree is a binary tree where all the nodes are inserted towards one side only.

- If all the nodes are inserted towards left subtree, the binary tree is said to be skewed towards left.
- If all the nodes are inserted towards right subtree, the binary tree is said to be skewed towards right.
- For example,



Left skewed binary tree

Right skewed binary tree

What is ADT binary tree?

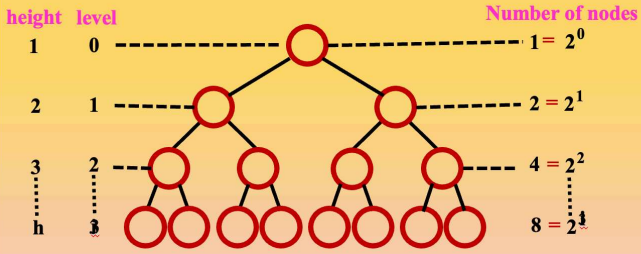
Definition: An Abstract data type binary tree in short called ADT binary tree is defined as

- Set of items (objects) along with type of each item to be stored in the tree and
- Set of various operations to be performed on those items (objects).
- The operations specified may be insert, delete, display tree contents, compare two trees etc.
- ADT Binary_Tree is
  - Objects : Finite set of nodes either empty or consisting of a node, left subtree and right subtree
  - Functions :
    - item : element to be inserted.
    - root : the root node of the binary tree.

NODE	insert (item, root)	::	Inserts an item into tree and returns the address of the root node
NODE	delete_item (item, root)	::	Deletes an item from the tree if found otherwise display "Item not found"
void	preorder (root)	::	Display tree in preorder if tree is not empty
void	inorder (root)	::	Display tree in inorder if tree is not empty
void	postorder (root)	::	Display tree in postorder if tree is not empty
int	count_nodes (root)	::	Returns number of nodes in the tree
int	height (root)	::	Returns height of the tree

How to find number of nodes at level i and height h?

The height of the tree can be computed as shown below:

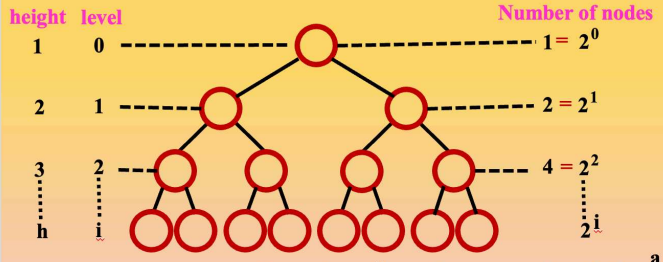


Maximum number of nodes at level i = 2^i

Maximum number of nodes at height h = 2^i

How to find total number of nodes at height h (or depth h)

The height of the tree can be computed as shown below:

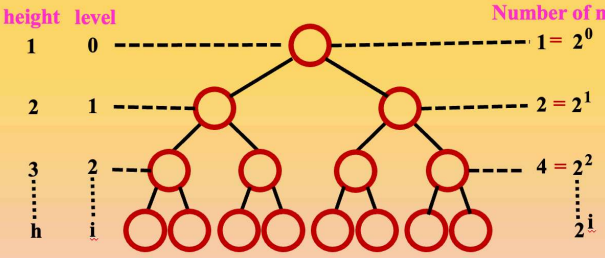


Total number of nodes $= 2^0 + 2^1 + 2^2 + 2^3 + \ldots\ldots 2^i$ G.P $= \frac{a\,(r^n - 1)}{r - 1}$

$$n = \frac{1\,(2^{i+1} - 1)}{2 - 1} = 2^{i+1} - 1$$

Total number of nodes $= 2^{i+1} - 1$

How to find height (or depth) of the tree?



Total number of nodes $= 2^0 + 2^1 + 2^2 + 2^3 + \ldots\ldots 2^i$

$$n = \frac{1\,(2^{i+1} - 1)}{2 - 1} = 2^{i+1} - 1$$

Total number of nodes $= 2^{i+1} - 1$

Height/Depth of the tree = h = max. level + 1 = i + 1

$n = 2^{i+1} - 1 = 2^h - 1$

$2^h = n + 1$

Taking log on both sides,

$\log(2^h) = \log(n + 1)$

$h \log_2(2) = \log_2(n + 1)$

$h = \log(n)$

So, height of the tree $h = \log_2(n)$

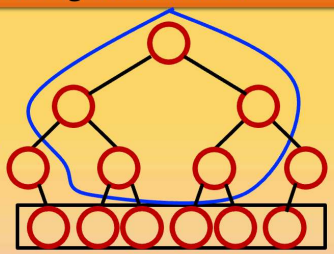
What is the relation between number of leaf nodes and degree-2 nodes?

Solution: The relation between number of leaf nodes and degree-2 nodes can be obtained as shown below:

- Degree – 0 nodes : Nodes with zero children = n_0 = 6
- Degree – 1 nodes : Nodes with one child = n_1 = 2
- Degree – 2 nodes : Nodes with two children = n_2 = 5
- Total number of nodes in the tree = $n = n_0 + n_1 + n_2$ ..... (1) 13
- Relationship between the number of nodes and number of branches

Total number of nodes = Total number of branches + 1

i.e., $n = B + 1$ ..... (2)



13 = 12 + 1

- For a node with degree – 0, number of branches = 0
  So, for n_0 number of nodes of degree – 0, number of branches = 0 ..... (3)
- For a node with degree – 1, number of branches = 1
  So, for n_1 number of nodes of degree – 1, number of branches = n_1 ..... (4)
- For a node with degree – 2, number of branches = 2
  So, for n_2 number of nodes of degree – 2, number of branches = $2n_2$ ..... (5)

Total number of nodes = $2n_2 + n_1 + 1$

Using eq. (1), $n_0 + n_1 + n_2 = 2n_2 + n_1 + 1$

$n_0 = n_2 + 1$

(Number of leaf nodes) (Number of degree – 2 nodes)

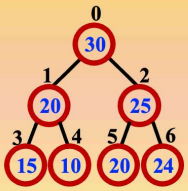
The different representations of binary trees are:

- Array representation
- Linked representation

How to represent a binary tree using arrays?

A binary tree can be represented using array representation as shown below:

- Start numbering the nodes of the tree from top to bottom i.e., level by level starting from 0
- In a specific level, number the nodes from left to right in sequence (in ascending order)
- The number of the left most node in a level must be one more than the highest number in previous level.
- These numbers represent the indices of an array and node values will be the corresponding array items as shown in figure below:



	0	1	2	3	4	5	6
A	30	20	25	15	10	20	24

How to compute maximum size of the array?

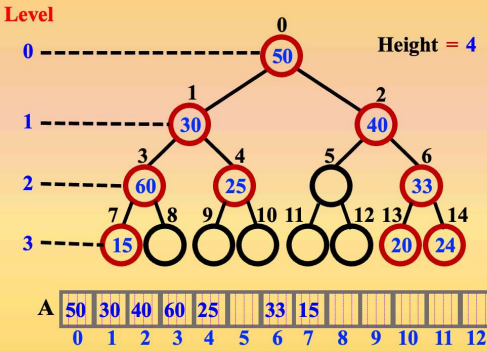
Observe the following points:

- Height of the tree = 3 h
- Total number of nodes = $2^h - 1 = 7$
- Size of the array = $2^h - 1 = 7$

How to represent a binary tree using arrays?

A binary tree can be represented using array representation as shown below:

- Start numbering the nodes of the tree from top to bottom i.e., level by level starting from 0
- In a specific level, number the nodes from left to right in sequence (in ascending order)
- The number of the left most node in a level must be one more than the highest number in previous level.
- These numbers represent the indices of an array and node values will be the corresponding array items as shown in figure below:



How to compute maximum size of the array?

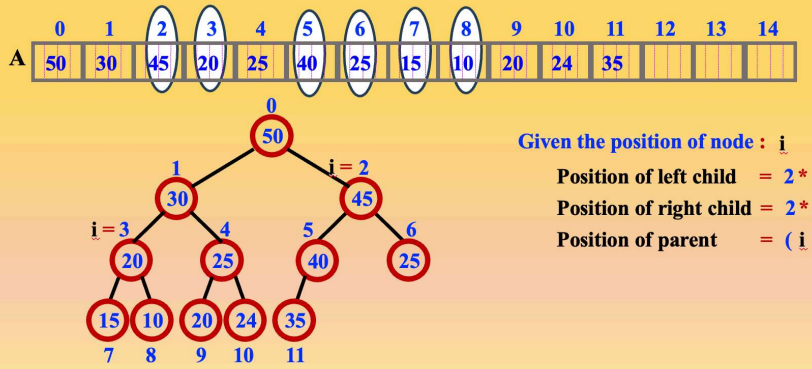
Observe the following points:

- Height of the tree = 3 h
- Total number of nodes = $2^h - 1 = 7$
- Size of the array = $2^h - 1 = 7$

Maximum size of the array = $2^4 - 1 = 15$

In the above tree black nodes are dummy nodes. They are not present.

How to find the left child, right child and parent of a given node in array representation?



Given the position of node : i

- Position of left child = 2 * i + 1
- Position of right child = 2 * i + 2
- Position of parent = (i - 1) / 2

What is linked representation of a binary tree?

In a linked representation of a binary tree can be represented as shown below:

- Each node has three fields:
  - info : contains the data to be manipulated
  - llink : contains the address of left subtree
  - rlink : contains the address of right subtree
- Structure definition for a node can be written using two methods:

Method 1:

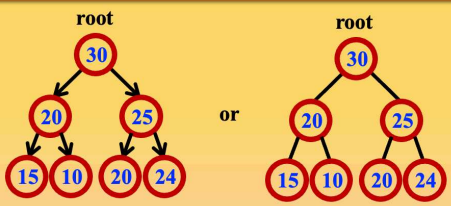
```
struct node
{
    int          info;
    struct node *llink;
    struct node *rlink;
};

typedef struct node * NODE;
```

Method 2:

```
typedef struct node * NODE;

struct node
{
    int     info;
    NODE    llink;
    NODE    rlink;
};
```

- The pictorial representation of a node can be written as shown below:

What is linked representation of a binary tree?




```
typedef struct node * NODE;

struct node
{
    int     info;
    NODE    llink;
    NODE    rlink;
};

struct node *root;
        OR
NODE root;
```

How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node

Linked representation

```
void preorder ( NODE root )
{
    if ( root == NULL) return;
    printf (" %d ", root -> info);
    preorder (root -> llink);
    preorder (root -> rlink);
}

void inorder( NODE root )
{
    if ( root == NULL) return;
    inorder (root -> llink);
    printf (" %d ", root -> info);
    inorder (root -> rlink);
}

void postorder ( NODE root )
{
    if ( root == NULL) return;
    postorder (root -> llink);
    postorder (root -> rlink);
    printf (" %d ", root -> info);
}
```


How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node

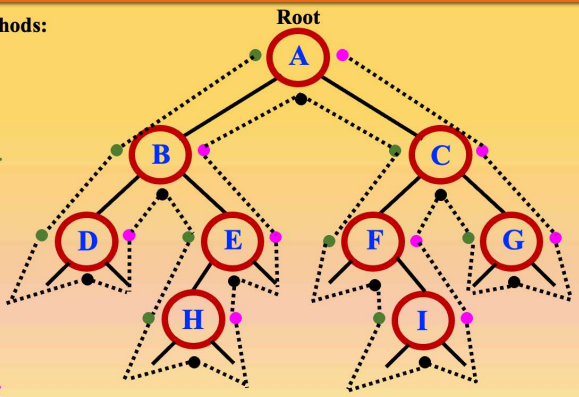
Array representation

```
void preorder ( int root[], int i )
{
    if ( root [ i ] == 0 ) return;
    printf (" %d ", root [ i ] );
    preorder (root , 2 * i + 1 );
    preorder (root , 2 * i + 2 );
}

void inorder ( int root[], int i )
{
    if ( root [ i ] == 0 ) return;
    inorder (root , 2 * i + 1 );
    printf (" %d ", root [ i ] );
    inorder (root , 2 * i + 2 );
}

void postorder ( int root[], int i )
{
    if ( root [ i ] == 0 ) return;
    postorder (root , 2 * i + 1 );
    postorder (root , 2 * i + 2 );
    printf (" %d ", root [ i ] );
}
```

How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node

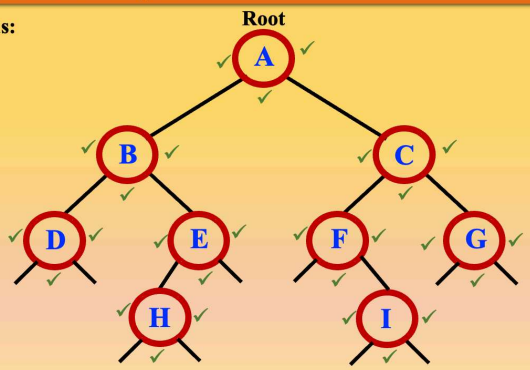


1. Preorder : A B D E H C F I G
2. Inorder : D B H E A F I C G
3. Postorder : D H E B I F G C A

How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node

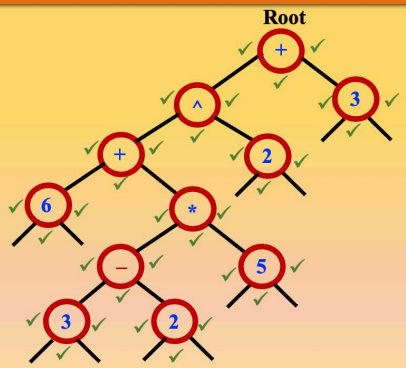


1. Preorder : A B D E H C F I G
2. Inorder : D B H E A F I C G
3. Postorder : D H E B I F G C A

How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node



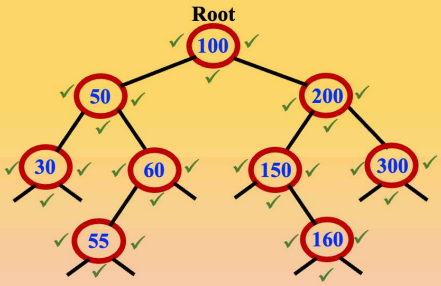
Infix : (6 + (3 – 2) * 5) ^ 2 + 3

1. Preorder : + ^ + 6 * - 3 2 5 2 3
2. Inorder : 6 + 3 - 2 * 5 ^ 2 + 3
3. Postorder : 6 3 2 - 5 * + 2 ^ 3 +

How to traverse the binary tree?

The tree can be traversed using following traversal methods:

- Preorder
  1. Visit the node
  2. Recursively traverse left subtree in Preorder
  3. Recursively traverse right subtree in Preorder
- Inorder
  1. Recursively traverse left subtree in Inorder
  2. Visit the node
  3. Recursively traverse right subtree in Inorder
- Postorder
  1. Recursively traverse left subtree in Postorder
  2. Recursively traverse right subtree in Postorder
  3. Visit the node



Tree sort

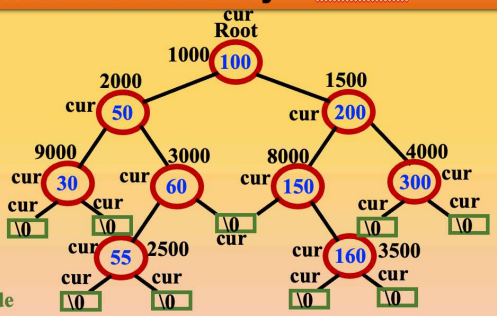
1. Preorder : 100 50 30 60 55 200 150 160 300
2. Inorder : 30 50 55 60 100 150 160 200 300
3. Postorder : 30 55 60 50 160 150 300 200 100

Dr. Padma Reddy A.M | Sai Vidya Institute of Technology | Bengaluru | download: nandipublications.com, saividya.ac.in

How to design a program to traverse the tree iteratively in inorder?

```
void inorder ( NODE root )
{
    NODE  cur,  stack[20] ;
    int       top  =  -1;

    if ( root == NULL)
    {
        printf ("List is empty\n");
        return;
    }

    cur  =  root;
    for ( ;; )
    {
        while (cur  !=  NULL)          // Reach leftmost node
        {
            s [++ top ] = cur;         // Push the node
            cur  =  cur -> llink;      // Traverse left subtree
        }

        if (top == -1) return;         // All nodes have been visited

        cur  =  s [ top -- ];          // Remove the node from stack
        printf (" %d ", cur -> info);  // Visit the node
        cur  =  cur -> rlink;          // Traverse right subtree
    }
}
```

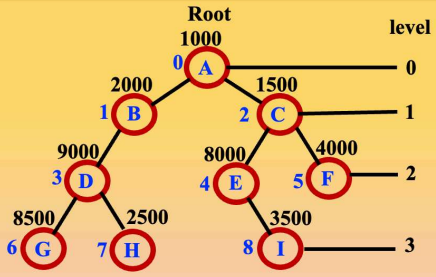
Stack: 4000, 3500, 8000, 1500, 2500, 3000, 9000, 2000, 1000



(2) Inorder : 30 50 55 60 100 150 160 200 300

What is level order traversal of a binary tree?

Definition: The nodes in a tree are numbered starting with root on level 0, continuing with the nodes on level 1, level 2 and so on. Nodes on any level are numbered from left to right. Visiting the nodes using the ordering suggested by node numbering is called level order traversal of a tree.

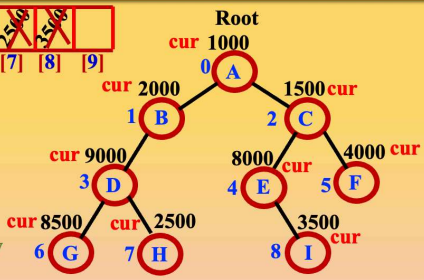


Level order : A B C D E F G H I

How to design a function for level order traversal of a binary tree?

```
void level_order ( NODE root )
{
    NODE  cur,  q[20] ;
    int       front, rear;

    if (root == NULL)
    {
        printf ("Tree is empty\n") ;
        return;
    }
    front  =  0,   rear  =  -1;
    q [ ++ rear ]  =  root;          // Insert root into queue
    while (front <= rear )           // As long as q is not empty
    {
        cur  =  queue [front++];     // Delete from queue

        printf (" %d ", cur -> info);// Visit the node

        if (cur -> llink  !=  NULL)  // Insert left child into q
            q [ ++ rear ] =  cur -> llink ;

        if (cur -> rlink  !=  NULL)  // Insert right child into q
            q [ ++ rear ] =  cur -> rlink;
    }
}   printf ( " \n " );
```

q: 1000 2000 1500 9000 8000 4000 8500 2500 3500 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

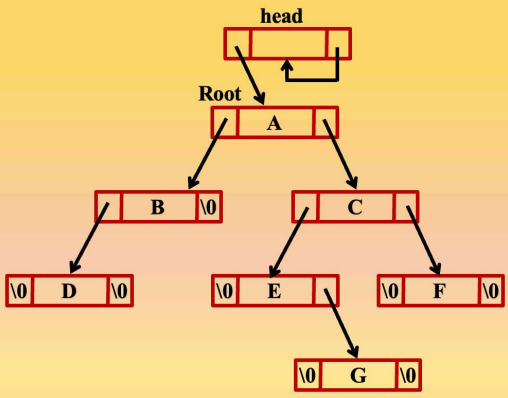


Level order : A B C D E F G H I

What are the disadvantages of binary trees?

Observe the following points:

- Total number of nodes = 7 = n
- Number of actual addresses = 6 = n - 1
- Number of NULL links = 8 = n + 1
- Total number of links = 14 = 2 n
- NULL links are more than the actual addresses
- There are n + 1 NULL links out of 2n total links



Disadvantages of binary trees

- Wasting memory simply by storing \0 characters
- Traversing a tree uses implicit stack in case of recursive traversal and uses explicit stack in case of iterative traversal. So, most of the time is spend in push and pop operations.
- Traversing a binary tree is time consuming.

Note: All the above disadvantages can be overcome using threaded binary trees.

What is threaded binary tree?

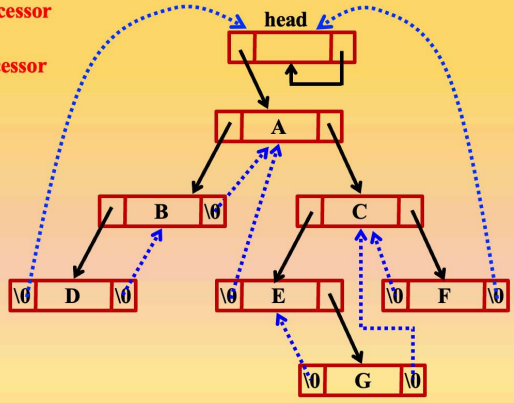
Definition: In a threaded binary tree, all the NULL links are replaced by actual addresses called threads.

- If left link of a node is NULL, replace it with its inorder predecessor if exists. Otherwise, replace it with address of header node.
- If right link of a node is NULL, replace it with its inorder successor if exists. Otherwise, replace it with address of header node.
- A binary tree where all the NULL links are replaced by actual addresses (either inorder predecessor or inorder successor or header node) is called a threaded binary tree.
- The structure can be defined as shown below:

```
struct   node
{
        int              info;

        struct   node   *llink;

        struct   node   *rlink;
};
typedef  struct  node *NODE;
```

- The header node can be declared as shown below:

```
struct node * head ;
        OR
NODE   head ;
```



Inorder traversal: D B A E G C F

How to represent a threaded binary tree in memory?

To represent a threaded binary tree in memory

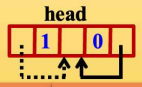
- We must be able to distinguish between normal link and a thread.
- It is done by adding two additional fields to the node structure :

  lthread : 1 – llink is a thread (denoted by dotted arrow)
  0 – llink is normal link (denoted by black arrow)

  rthread : 1 – rlink is a thread (denoted by dotted arrow)
  0 – rlink is normal link (denoted by black arrow)

```
struct node
{
        int             info;
        short int       lthread;
        struct node     *llink;
        short int       rthread;
        struct node     *rlink;
};
typedef struct node *NODE;
```

- An empty tree can be represented as shown below:



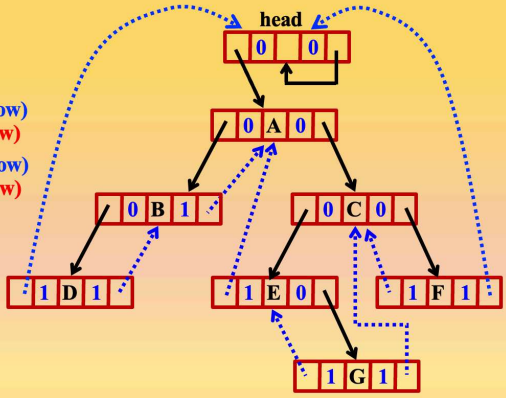


Fig. Memory representation of a threaded binary tree

Inorder traversal: D B A E G C F

How to traverse a threaded binary tree?

The function to find inorder successor can be written as shown below:

```
NODE inorder_successor (NODE x )
{
    NODE    cur ;
    cur = x-> rlink;  // Get the address of right node
    if ( x -> rthread == 1) return cur;
    // Keep moving left till you get thread in left link
    while ( cur -> lthread == 0) cur = cur -> llink;
    return cur;
}
```

The function to traverse the tree in inorder is shown below:

```
void inorder (NODE head )
{
    NODE cur;
    cur = head;
    for ( ;; )
    {
        cur = inorder_successor (cur );
        if ( cur == head ) return;
        printf ("%d ", cur ->info );
    }
}
```

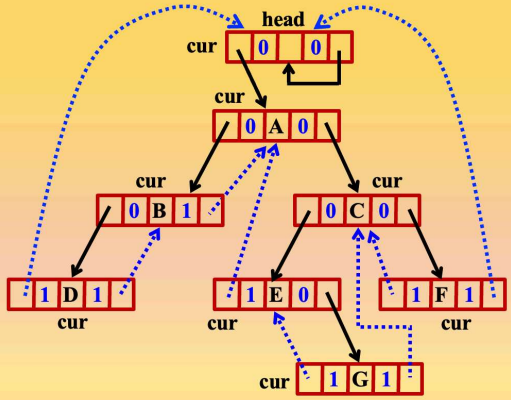


Fig. Memory representation of a threaded binary tree

Inorder traversal: D B A E G C F
✓ ✓ ✓ ✓ ✓ ✓ ✓