



# Data Structure with C

Sub Code: 06CS34

Solved question papers – **Dec 07 / Jan 08 (New Scheme)**

Contributed by: Megha B.R, BMSCE, Bangalore

Reviewed by: Mr. Keshav Murthy, Asst. Professor, BIT  
Bangalore

Please send us feedbacks about this solved paper at [admin.tb3@enggresources.com](mailto:admin.tb3@enggresources.com)

Copyright – tB3

Note: Publishing this paper in any form for commercial use without written permission from tB3 is strictly prohibited, failing which can lead to prosecution under copyright act of Govt. of India.

Part A

1.a. Given the following declarations: 05

int x; double d; int \*p; double \*q;

which of the following expressions are not allowed?

- i) p = &x;      ans: allowed.
- ii) P = &d;      ans: not allowed (type mismatch)
- iii) q = &x;      ans: not allowed (type mismatch)
- iv) q = &d;      ans: allowed
- v) p =x;      ans: not allowed(a variable value cannot be assigned to a pointer)

b. Show what would be printed from the following block: 06

```
/* Local definitions */
```

```
int x[2][3] = {  
                {4,5,2},  
                {7,6,9}  
            };
```

```
/* Statements */
```

```
fun(x);  
fun(x+1);  
return 0;
```

```
void fun(int (*p)[3])  
{  
    printf("\n %d%d",(*p)[0],(*p)[1],(*p)[2]);  
    return;  
}
```

ans: Here, p is defined as a pointer to a group of one dimensional arrays, each having 3 elements in a row.

So, x refers to the first row and (x+1) refers to the second row.

Accordingly, the output of the above program is

```
4 5 2  
7 6 9
```

c. Briefly explain memory allocation functions. 09

soln: Dynamic memory allocation is the process of allocating memory during execution time(run time). This allocation technique uses predefined functions to allocate and release memory for data during execution time. So if there is an unpredictable storage requirement, then the dynamic memory allocation technique is used.

Three functions that support dynamic memory allocation are:

1. malloc(size) -

This function allows the program to allocate memory explicitly as and when required and the exact amount needed during execution.. This function allocates a block of memory, the size of which is the number of bytes specified in the parameter. The syntax is :

```
ptr = (datatype *) malloc (size);
```

or

```
ptr=(datatype *)malloc(n*sizeof(datatype));
```

On successful allocation, the function returns the address of first byte of allocated memory.

If specified size of memory is not available, the function returns NULL.

2. calloc(n,size)-

This function is used to allocate multiple blocks of memory.The number of blocks is determined by the first parameter n. The size of each block is equal to the number of bytes specified in the parameter ie.,size. Thus the total number of bytes allocated is n\*size and all bytes will be initialized to 0. The syntax is :

```
ptr = (datatype*)calloc(n,size);
```

Here, n is the number of blocks to be allocated.

size is the number of bytes in each block.

3. realloc(ptr,size)-

Sometimes, the allocated memory may not be sufficient and we may require additional memory space. Also sometimes, the allocated memory may be much larger and we want to reduce the size of allocated memory. In both situations, the size of allocated memory can be changed using realloc() and the process is called reallocation of memory.

```
Ptr = (data_type*)realloc(ptr,size);
```

Here, ptr is the pointer to a block of previously allocated memory.

Size is new size of the block.

2.a. Implement i) Copying one string to another ii)Reverse the given string without using string library functions in C.

12

```
soln: void newstrcpy(char str[], char newstr[])
```

```
{
```

```
    int i;
```

```
    for(i=0 ; str[i] != '\0' ; i++)
```

```
        newstr[i] = str[i];
```

```
    newstr[i] = '\0';
```

```
}
```

```
void newstrrev(char str[] , char rev[])
```

```
{
```

```
    int i,n;
```

```

/* find the length of the string */
for(i=0; str[i] != '\0' ; i++);
n = i;

/*reverse the string */
for(i=0; str[i] != '\0' ; i++)
    rev[n-1-i] = str[i];
rev[n] = '\0';
}

```

b. Write a C program to represent a complex number using structure and add 2 complex numbers.

08

soln:

```

#include<stdio.h>
#include<conio.h>

typedef struct
{
    float r;    // real part
    float i;    // imaginary part
}COMPLEX;

COMPLEX add(COMPLEX a, COMPLEX b)
{
    COMPLEX c;

    c.r = a.r + b.r ;
    c.i = a.i + b.i ;

    return c;
}

COMPLEX read()
{
    COMPLEX x;

    printf("Real part : ");
    scanf("%d",&x.r);
    printf("Imaginary part : ");
    scanf("%d",&x.i);

    return x;
}

```

```

void write(COMPLEX a)
{
    if(a.i >= 0)
        printf(“%5.2f + %5.2fi\n”,a.r,a.i);
    else
        printf(“%5.2f%5.2fi\n”,a.r,a.i);
}

void main()
{
    COMPLEX a , b , c;
    clrscr();

    printf(“Enter the first complex number\n”);
    a = read();

    printf(“Enter the second complex number\n”);
    b = read();

    c = add( a , b );

    printf(“Sum of complex numbers = “);
    write(c);
    getch();
}

```

3.a. Define a stack and operations over a stack. Implement reversing a string using a stack in C. 12

**soln:** A stack is a special type of a data structure where elements are inserted from one end and elements are deleted from the same end. This position from where elements are inserted and from where elements are deleted is called top of stack. Using this approach, the last element inserted is the first element to be deleted out, and hence stack is also called Last In First Out(LIFO) data structure.

The various operations that can be performed on stacks are :

- push – Inserting an element on top of stack
- pop – Deleting an element from the top of stack
- display – Display contents of stack

/\* Reverse a string using stack \*/

```

#include<stdio.h>

void str_rev(char a[])
{
    char s[10];
    int i,top;

    /* Push all the characters onto stack */
    top = -1;
    for(i=0;a[i] != '\0'; i++)
    {
        s[++top] = a[i];
    }

    /* Pop from stack and store it in a given array */
    i = 0;
    while(top != -1)
    {
        a[i++] = s[top--];
    }
}

void main()
{
    char a[10];

    printf("Enter the string\n");
    scanf("%s",a);

    str_rev(a);

    printf("Reversed string = %s\n",a);
}

```

b. What is recursion? Explain efficiency of recursion. Write a C recursive program to solve tower of hanoi problem. 08

soln: Recursion is a technique for defining a problem in terms of one or more versions of the same problem. A function, which contains a call to itself or a call to another function, which eventually causes the first function to be called, is known as a recursive function.

The efficiency of recursion can be understood by its advantages:

- Clearer and simpler versions of recursive functions can be created.
- Recursive definition can be easily translated into a recursive function.
- Lot of “bookkeeping” activities such as intialization etc., required in iterative

solution can be avoided.

- Many functions are easier to implement recursively and are very efficient.

```
/* Tower Of Hanoi */

#include<stdio.h>

void tower(int n, char source, char temp, char destination)
{
    if(n == 1)
    {
        printf("Move disc %d from %c to %c\n",n,source,destination);
        return;
    }

    /* Move n-1 discs from source to temp */
    tower(n-1 , source, destination , temp);

    /* Move nth disc from source to destination */
    printf("Move disc %d from %c to %c\n",n,source,destination);

    /* Move n-1 discs from temp to destination */
    tower(n-1 , temp , source , destination);
}

void main()
{
    int n;

    printf("Enter the number of discs\n");
    scanf("%d",&n);

    /* Transfer n discs from needle A to needle C*/
    tower(n , 'S' , 'T' , 'D');
}
```

4.a. Write a C program to implement multiple stacks using single array.

12

soln: This is implemented using array of singly linked lists as shown below:

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

```
typedef struct node
```

```

{
    int info;
    struct node *link;
} *NODE;

/* Function to get a node */
NODE getnode()
{
    NODE x;
    x = (NODE)malloc(sizeof(struct node));
    if(x==NULL)
    {
        printf("Out of memory\n");
        exit(0);
    }
    return x;
}

/* Function to insert a node */
NODE insert_front(int item , NODE first )
{
    NODE temp;
    temp = getnode();
    temp->info = item;
    temp->link = first;
    return temp;
}

/* Function to delete a node */
NODE delete_front(NODE first)
{
    NODE temp;
    if(first == NULL)
    {
        printf("List is empty, cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->link;
    printf("Item deleted = %d\n",first->info);
    free(first);
    return temp;
}

/* Function to display all the stacks */

```

```

void display(NODE a[],int n)
{
    NODE temp;
    int i;

    printf("Contents of multiple stacks \n");

    for(i=0; i<n; i++)
    {
        printf("Stack %d\n",i);
        if(a[i] == NULL)
        {
            printf("Empty\n");
            continue;
        }
        temp = a[i];
        while(temp!= NULL)
        {
            printf("%d ",temp->info);
            temp = temp->link;
        }
        printf("\n");
    }
}

void main()
{
    int item, choice, n , i;
    NODE a[5] = {NULL};
    printf("Enter number of stacks\n");
    scanf("%d",&n);
    while(1)
    {
        printf("1. Insert\n 2. Delete\n 3. Display\n 4. Exit\n");
        printf("Enter the choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("Enter stack number and item\n");
                    scanf("%d%d",&i,&item);
                    a[i] = insert_front(item,a[i]);
                    break;

            case 2: printf("From which stack number to delete?\n");
                    scanf("%d",&i);

```

```

        printf("Stack %d : ",i);
        a[i] = delete_front(a[i]);
        break;

    case 3: display(a,n);
            break;

    default: exit(0);
}
}
}

```

b. What is a linear queue? What are the applications of linear queue? Implement insert and delete operations.

08

soln: A queue is defined as a special type of data structure where elements are inserted from one end and elements are deleted from another end. The end from where the elements are inserted is called rear end and the end from where elements are deleted is called front end. Since the first item inserted is the first item to be deleted from queue, queue is also called First In First Out (FIFO) data structure.

Queues are useful in time sharing systems where many user jobs will be waiting in the system queue for processing. These jobs may request the services of the CPU, main memory or external devices such as printer etc. All these jobs will be given a fixed time for processing and are allowed to use one after the other. This is the case of an ordinary queue where priority is the same for all the jobs and whichever job is submitted first, that job will be processed.

/\* Implementation of insert and delete operations using global variables\*/

```

void insert_rear()
{
    int item;

    /*Check for overflow of queue */
    if(r == queue_size-1)
    {
        printf("Queue overflow\n");
        return;
    }

    printf("Enter the item\n");
    scanf("%d",&item);
    r = r+1;
}

```

```

        q[r] = item;
    }

void delete_front()
{
    if(f>r)
    {
        printf("Queue underflow\n");
        return;
    }

    printf("The element deleted = %d\n",q[f]);
    f++;

    if(f>r)
    {
        f = 0;        /* Initialize to queue empty conditions */
        r = -1;
    }
}

```

5.a. Given an ordered linked list whose first node is denoted by 'start' and node is represented by 'key' as information and 'link' as link field. Write a C program to implement deleting number of nodes (consecutive) whose 'key' values are greater than or equal to 'Kmin' and less than 'Kmax'.

12

Soln:

```

#include<stdio.h>
#include<conio.h>

```

```

struct node
{
    int key;
    struct node *link;
};
typedef struct node *NODE;

```

```

NODE delete_kmin_to_kmax(NODE first, int kmin, int kmax)
{
    NODE cur,prev;

    while(1)
    {

```

```

    if(first == NULL)
        return NULL;

    if(kmin<=first->key && first->key<=kmax)
    {
        cur = first;
        first = first->link;
        free(cur);
        continue;
    }

    prev = NULL;
    cur = first;

    while(cur != NULL)
    {
        if(kmin<=cur->key && cur->key<=kmax)
            break;
        prev = cur;
        cur = cur->link;
    }
    if(cur == NULL)
        return first;

    prev->link = cur->link;
    free(cur);
    continue;
}
}

```

/\* function to create an ordered linked list\*/

NODE insert (int item, NODE first)

```

{
    NODE temp,prev,cur;

    temp = (NODE)malloc(sizeof(struct node));
    temp->info = item;
    temp->link = NULL;

    if(first == NULL) return temp;

    if(item < first->info)
    {
        temp->link = first;
        return temp;
    }
}

```

```

}
prev = NULL;
cur = first;

while(cur!= NULL && item >cur->info)
{
    prev = cur;
    cur = cur->link;
}
prev->link = temp;
temp->link = cur;

return first;
}

void main()
{
    NODE first = NULL;
    int choice, item,min,max;
    clrscr();
    while(1)
    {
        printf("1.Insert 2.Delete 3.Exit\n");
        printf("Enter the choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("Enter the item to be inserted\n");
                    scanf("%d",&item);
                    first = insert(item,first);
                    break;
            case 2: printf("Enter the min and max values\n");
                    scanf("%d%d",&min,&max);
                    first = delete_kmin_to_kmax(first,min,max);
                    break;
            default: exit(0);
        }
    }
}

```

b. Write a C program to implement insertion to the immediate left of kth node in the list.

08

soln:

```
NODE insert_left(int item, int pos, NODE first)
{
    NODE temp;    // Node to be inserted
    NODE prev,cur; // Insert temp between these 2 nodes
    int count;    // To find the position to insert

    temp = (NODE)malloc(sizeof(struct node));
    temp->info = item;
    temp->link = NULL;

    if(first == NULL || pos-1 == 1) // Insert the node for the first time
    {
        return temp;
    }

    if( first == NULL)
    {
        printf("Invalid position\n");
        return first;
    }

    if(pos-1 == 0)
    {
        temp->link = first;
        return temp;
    }

    /* Find the appropriate position */
    count = 1;
    prev = NULL;
    cur = first;

    while(cur!=NULL && count != pos-1)
    {
        prev = cur;
        cur = cur->link;
        count++;
    }

    if(count == pos-1)
    {
        prev->link = temp;
        temp->link = cur;
    }
}
```

```

        return first;
    }

    printf("Invalid position\n");
    return first;
}

```

6.a. Write a C program to implement doubly linked list with following operations:

i) create ii) insert.

10

soln:

```

#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>

```

```

struct node()
{
    int info;
    struct node *rlink;
    struct node *llink;
};
typedef struct node *NODE;

```

```

NODE create(NODE head)

```

```

{
    NODE temp,cur;
    int item;
    char ch = 'y';

    while(ch == 'y')
    {
        temp = (NODE)malloc(sizeof(struct node));
        printf("Enter the item\n");
        scanf("%d",&temp->info);

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

        head->llink = temp;
        temp->rlink = head;
        temp->llink = cur;
        cur->rlink = temp;

        printf("Do you want to enter another node? (y\n)n");
        scanf("%c",&ch);
    }
}

```

```

    }
    return head;
}

NODE insert_front(int item, NODE head)
{
    NODE temp , cur;

    temp = (NODE)malloc(sizeof(struct node));
    temp->info = item;

    cur = head->rlink; // obtain address of first node

    head->rlink = temp;
    temp->llink = head;
    temp->rlink = cur;
    cur->llink = temp;

    return head;
}

```

```

void main()
{
    NODE head;
    char choice;
    int item;

    head= (NODE)malloc(sizeof(struct node));
    head->rlink = head;
    head->llink = head;

    head = create(head);

    while(1)
    {
        printf("Do you want to insert a node at the front end?\n");
        scanf("%c",&choice);
        if(choice == 'y')
        {
            printf("Enter the item\n");
            scanf("%d",&item);
            head = insert_front(item,head);
        }
        else
            break;
    }
}

```

```

    }
}

```

b. Implement concatenation of 2 circular singly linked lists List1 and List2. Use header nodes to implement the list. 10

soln: Suppose two circular lists are created with header nodes identified by list1 and list2. The following function concatenates the 2 lists:

```

NODE concatenate(NODE list1, NODE list2)
{
    NODE cur1,cur2;
    /* Obtain the address of the last node of the first list */
    cur1 = list1->link;
    while(cur1->link != list1)
        cur1 = cur1->link;

    /* Obtain the address of the last node of the second list */
    cur2 = list2->link;
    while(cur2->link != list2)
        cur2 = cur2->link;

    /* Attach the first node of list2 to end of list1 */
    cur1->link =list2->link;

    /* Point the last node of list2 to header of list1 */
    cur2->link = list1;

    return list1;
}

```

7.a. Implement binary tree traversals in C: i) Inorder ii) Preorder iii)Postorder. 10

soln:

```

void inorder(NODE root)
{
    if(root == NULL)
        return;

    inorder(root->llink);
    printf("%d",root->info);
}

```

```

        inorder(root->rlink);
    }

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

void postorder(NODE root)
{
    if(root == NULL)
        return;

    postorder(root->llink);
    postorder(root->rlink);
    printf("%d",root->info);
}

```

b. What are the applications of binary tree? Implement binary search tree and check for duplicate data.

10

soln: Applications of binary search trees are :

- Searching and sorting.
- Manipulation of arithmetic expression.
- Constructing symbol table.
- The trees are also used in syntax analysis of compiler design and are used to display the structure of a sentence in a language.

/\* Function to insert an item into a binary search tree( No duplicate items are allowed)\*/

```

NODE insert(int item, NODE root)
{
    NODE temp , cur, prev;

    temp = (NODE)malloc(sizeof(struct node));

```

```

temp->info = item;
temp->llink = NULL;
temp->rlink = NULL;

if(root == NULL)
    return temp;
prev = NULL;
cur = root;
while(cur!=NULL)
{
    prev = cur;
    if(item == cur->info)
    {
        printf("Duplicate entries not allowed\n");
        free(temp);
        return root;
    }

    if(item < cur->info)
        cur = cur->llink;
    else
        cur = cur->rlink;
}
if(item < prev->info)
    prev->llink = temp;
else
    prev->rlink = temp;

return root;
}

```

8. Write short notes on:

20

- a) Threaded binary trees.
- b) Applications of stack.
- c) Array implementation of binary trees.
- d) Structures and union.

Soln:

a) Threaded binary tree – The link fields in a binary tree which contain NULL values can be replaced by address of some nodes in the tree which facilitate upward movement in the tree. These extra links which contain addresses of some nodes are called threads and the tree is termed as threaded binary tree.

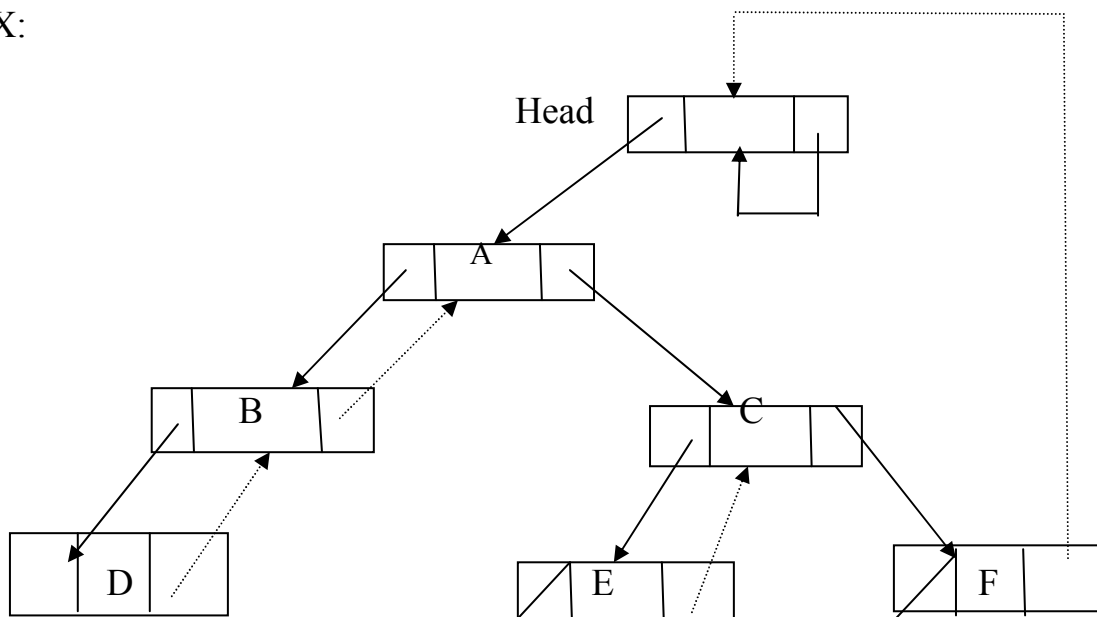
There are three types of threaded binary trees -

1. In-threaded binary tree - In a binary tree, if llink of any node contains null

and if it is replaced by address of inorder predecessor, then the resulting tree is called left inthreaded binary tree. If rlink of a node is null and if it is replaced by address of inorder successor, the resulting tree is called right inthreaded tree. A inthreaded binary tree is one which is both left and right in-threaded.

2. Pos-threaded binary trees - In a binary tree, if llink of any node contains null and if it is replaced by address of postorder predecessor, then the resulting tree is called left post threaded binary tree. If rlink of a node is null and if it is replaced by address of post order successor, the resulting tree is called right post threaded tree. A post-threaded binary tree is one which is both left and right post-threaded.
3. Pre-threaded binary trees – In a binary tree, if llink of any node contains null and if it is replaced by address of preorder predecessor, then the resulting tree is called left pre threaded binary tree. If rlink of a node is null and if it is replaced by address of preorder successor, the resulting tree is called right prethreaded tree. A prethreaded binary tree is one which is both left and right pre threaded.

EX:



Advantages of threaded binary trees -

- In a binary tree, more than 50% of space is wasted in storing null values. This wastage of memory space is avoided by storing addresses of some nodes.
- Traversing of a threaded binary tree is very fast. This is because, it does not use implicit or explicit stack.
- Computations of predecessor and successor of given nodes is very easy and efficient.

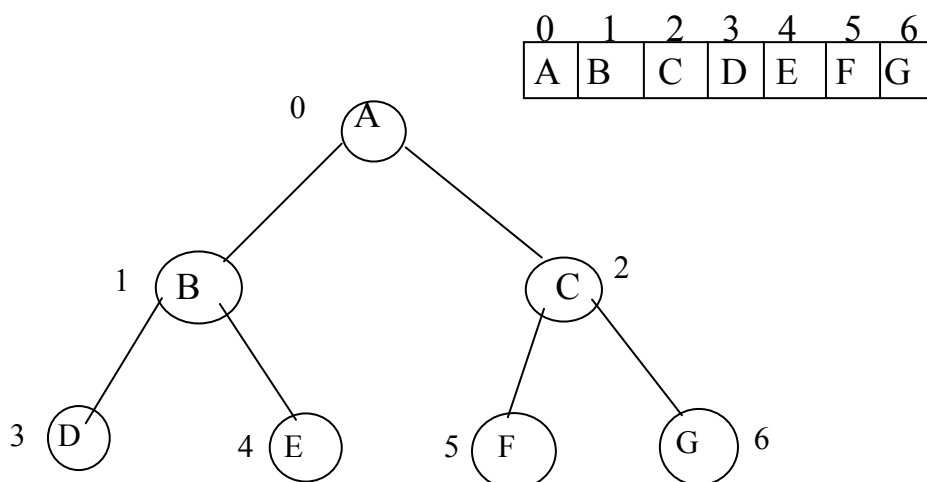
- Any node can be accessed from any other node. Using threads, upward movement is possible and using links downward movement is possible. Thus, in a threaded binary tree, we can move in either directions.
- Even though insertion into a threaded binary tree and deletion from a threaded binary tree are time consuming operations, they are very easy to implement.

b. Applications of stack -

- **Conversions of expressions** - When we write mathematical expressions in a program, we use infix expressions.. these expressions will be converted into equivalent machine instructions by the compiler using stacks. Using stacks we can efficiently convert the expressions from infix to postfix, infix to prefix, postfix to infix, postfix to prefix, postfix to infix, and postfix to prefix.
- **Evaluation of expressions** – An arithmetic expression represented in the form of either postfix or prefix can be easily evaluated using stacks.
- **Recursion** – A function which calls itself is called recursive function. Some of the problems such as Tower of Hanoi, problems involving tree manipulations etc., can be implemented very efficiently using recursion.
- **Other applications** – There are so many other applications where stacks can be used. : for example, to find whether the string is palindrome, to check whether a given expression is valid or not and so on.

c) Array implementation of binary trees.

Soln: A tree can be represented using an array, called sequential representation, as shown below :



We can observe the following points -

- The nodes are numbered sequentially from 0.
- The node with position 0 is considered as the root node.

- Given the position of any node  $i$ ,  $2i+1$  gives the position of the left child and  $2i+2$  gives the position of the right child.
- Given the position of any node  $i$ , the parent position is given by  $(i-1)/2$ . If  $i$  is odd, it points to the left child otherwise, it points to the right child.
- In practice, one can initialize each location in the array to 0 indicating the node is not used. Non-zero value in the location indicates the presence of the node.

D) Structures and union : A structure is defined as a collection of logically related variables under a single name. All these variables may contain data items of similar or dissimilar data types.

A union is a derived data type like structure. So, union can also be treated as a collection of variables under a single name. All these variables may contain data items of similar or dissimilar data types. Each variable in the union is called a member or a field.

### STRUCTURE

### UNION

1. The keyword struct is used to define a structure.	1. The keyword union is used to define a union.
2. The size of the structure is greater or equal to the sum of the sizes of its members.	2. The size of the union is equal to the size of the largest member.
3. Each member within a structure is assigned a unique storage area.	3. Memory allocated is shared by individual members of the union.
4. Individual members can be accessed at a time.	4. Only one member can be accessed at a time.
5. Altering the value of a member will not affect other members of the structure.	5. Altering the value of any of the member will alter other member values.

