Biyani's Think Tank

Concept based notes

Data Structure

(MCA)

Asst. Prof.
Nitasha Jain
Biyani Institute of Science and Management, Jaipur

Biyani's Group of Girls' Colleges

Published by :
Think Tanks
Biyani Group of Colleges

Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

Edition: 2011
New Edition: 2012

Leaser Type Setted by:
Biyani College Printing Department

Syllabus Data Structure

DATA STRUCTURE: Basic data structures such as arrays, stack and queues and their applications, linked and sequential representation. Linked list, representation of linked list, multi linked structures. Trees: definitions and basic concepts, linked tree representation, representations in contiguous storage, binary trees, binary tree traversal, searching insertion and deletion in binary trees, heap tree and heap sort algorithm, AVL trees.

Graphs and their application, sequential and linked representation of graph – adjacency matrix, operations on graph, traversing a graph, Dijkstra's algorithm for shortest distance, DFS and BFS, Hashing. Searching and sorting, use of various data structures for searching and sorting, Linear and Binary search, Insertion sort, Selection sort, Merge sort, Radix sort, Bubble sort, Quick sort, Heap Sort.

Reference Books

1. Data Structures in C/C++, Tanenbaum, PHI
2. Data Structures in C/C++, Horowitz, Sawhney.

DATA STRUCTURE: - INTRODUCTION

Q 1. What is Data Structure?

Ans. A data structure is an arrangement of data in a computer's memory or even disk storage. It has a different way of storing and organizing data in a computer so that it can used efficiently. Different kind of Data structure is used in different application. For example, B-tree is widely used in implementation of databases.

Q 2. Name different type of data structures?

Ans. There are basically two types of data structures

1 Linear data Structure

2 Non Linear data Structure

Q 3. What is Linear data Structure?

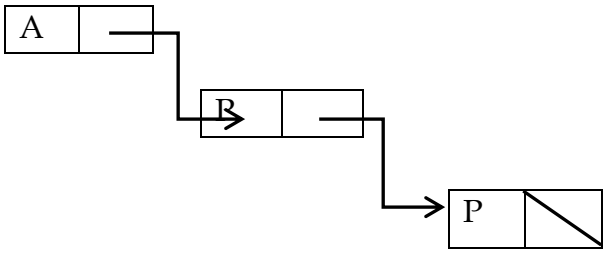
Ans. Linear data structure organize there elements in a linear fashion, where data elements are attached one after another. They are organized in a way similar to how the computer's memory is organized.

There are 2 ways of representing linear structure in memory:-

1) To have the linear relationship between the elements represented by means of sequential memory locations. For Example :- ARRAY

A	P	P	L	E
101	102	103	104	105

2) To have the linear relationship between the elements represented by means of pointers and links. For Example:- LINKED-LIST



Q 4. What is Non-Linear Data Structure?

Ans. Non Linear Data Structures are constructed by attaching a data element to several other data elements in such a way that it reflects a specific relationship among them

For Example :- N-dimensional Array, Trees And Graphs

ARRAYS

Q 1. What is an Array?

Ans. An Array can be defined as a data structure having fixed size sequenced collection of elements of same data type and which can be referenced through same variable name. An Array is a name given to a group of similar quantities.

Declaring an array

Type variable_name [length of array];

For example:-

double height[10];

float width[20];

int min[9];

int myArray[5] = {1, 2, 3, 4, 5}; //declare and initialize the array in one statement
In C language array starts at position 0. C language treats array name as the pointer to the first element and an iten in the array can be accessed through its index.

Q 2. How to read and write data in an array?

Ans. #include<stdio.h>

```
void main ()
{
 int array[10], n;

// Reading an array
printf("Enter size of an array");
 scanf ("%d", &n);
printf("Enter array elements");
for (i=0;i<n;i++)
scanf ("%d", &array[i]);
```

```
 // writing an array
 printf(“array elements are”);
for (i=0;i<n;i++)
printf ("%d", array[i]);
}
```

Q 3. Write code to insert data in an array at a particular position

Ans.

```
#include<stdio.h>
Void main ()
{
 int array [10], n elem, pos;

 // Reading an array
 printf (“Enter size of an array”);
scanf ("%d", &n);
printf (“Enter array elements”);
for (i=0;i<n;i++)
scanf ("%d", &array[i]);

//Inserting an element at specified position
printf (“Enter  element to be inserted”);
scanf ("%d", &elem);
printf(“Enter position where to insert the element”);
scanf ("%d", &pos);
 pos--;
for (i<n;i>=pos;i--)
a[i+1]=a[i];
a[pos]=elem;
n++;

// writing an array
printf (“array elements are”);
for (i=0;i<n;i++)
printf ("%d", array[i]);
}
```

Q 4. Write code to delete data from an array from a particular position?

Ans.

```
#include<stdio.h>
void main ()
{
 int array[10], n elem ,pos;

// Reading an array
printf("Enter size of an array");
scanf ("%d", &n);
printf("Enter array elements");
for (i=0;i<n;i++)
scanf ("%d", &array[i]);

//deleting  an element at specified position
printf("Enter position from where to delete the element");
 scanf ("%d", &pos);
 pos--;
for (i=pos ; i<n ;i++)
a[i]= a[i+1];
n--;

 // writing an array
 printf("array elements are");
for (i=0;i<n;i++)
printf ("%d", array[i]);
}
```

Q 5. Write code to merge two sorted arrays ?

Ans.

```
/* Program to merge two sorted arrays */
#include<stdio.h>
int merge(int[],int[],int[],int,int);
void main()
{
 int a[20],b[20],c[40],n,m,i,p;
 printf("\nEnter the no.of element present in the first array: ");
 scanf("%d",&n);
 printf("\nEnter the first array…….\n");
```

```
        for(i=0;i<n;i++)
            scanf("%d",&a[i]);
        printf("\nEnter the no. of element present in the second array: ");
        scanf("%d",&m);
        printf("\nEnter the second array…….\n");
        for(i=0;i<m;i++)
           scanf("%d",&b[i]);
        p=merge(a,b,c,n,m);
        printf("\nThe merged array is……\n");
        for(i=0;i<p;i++)
          printf("%d  ",c[i]);
          printf("\n\n");
}
int merge(int a[],int b[],int c[],int n,int m)
{
       int i=0,j=0,k=0;
       while(i<n&&j<m)
       {
            if(a[i]<b[j])
           {
                c[k]=a[i];
                 i++;
        k++;
  }
  else if(a[i]>b[j])
  {
        c[k]=b[j];
        j++;
        k++;
   }
   else          // to avoid duplication
   {
       c[k]=a[i];
       i++;
       j++;
              k++;
          }
          }
              for(;i<n;i++,k++)
              c[k]=a[i];
                for(;j<m;j++,k++)
```

```
                c[k]=b[j];
                return k;
}
```

Q 6. How to read and write data in 2-D array?

Ans.

```
#include <stdio.h>

void main()
{
  int a[3][2];
  int i,j;
for(i = 0;i<3;i++){
   for(j=0;j<2 ;j++) {
 a[i][j]=2;
}
  }

for(i = 0;i<3;i++){
for(j=0;j<2;j++) {
printf("value in array %d\n",a[i][j]);
}
}

 for(i = 0;i<3;i++){
 for(j=0;j<2;j++){
printf("value in array %d and address is %8u\n", a[i][j],&a[i][j]);
      }
   }
}
```

Q 7. Write an algorithm to multiply two 2-Dimensional array?

Ans.

```
#include<stdio.h>
void main()
{
int a[5][5],b[5][5],c[5][5],i,j,k,sum=0,m,n,o,p;
printf("\nEnter the row and column of first  matrix");
 scanf("%d %d",&m,&n);
```

```
 printf("\nEnter the row and column of second matrix");
scanf("%d %d",&o,&p);
if(n!=o){
printf("Matrix mutiplication is not possible");
printf("\nColumn of first matrix must be same as row of second matrix");
}
 else{
printf("\nEnter the First matrix->");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("\nEnter the Second matrix->");
 for(i=0;i<o;i++)
 for(j=0;j<p;j++)
scanf("%d",&b[i][j]);
printf("\nThe First matrix is\n");
 for(i=0;i<m;i++){
printf("\n");
 for(j=0;j<n;j++){
printf("%d\t",a[i][j]);
 }
 }
 printf("\nThe Second matrix is\n");
 for(i=0;i<o;i++){
 printf("\n");
for(j=0;j<p;j++){
printf("%d\t",b[i][j]);
 }
}
for(i=0;i<m;i++)
 for(j=0;j<p;j++)
c[i][j]=0;
for(i=0;i<m;i++){ //row of first matrix
 for(j=0;j<p;j++){  //column of second matrix
sum=0;
for(k=0;k<n;k++)
   sum=sum+a[i][k]*b[k][j];
c[i][j]=sum;
}
}
  }
```

```
 printf("\nThe multiplication of two matrix is\n");
 for(i=0;i<m;i++){
    printf("\n");
    for(j=0;j<p;j++){
        printf("%d\t",c[i][j]);
    }
 }
getch();
}
```

SEARCHING

Q 1. How to do Linear Search in an Array?

Ans. Linear search or Sequential is the simplest search algorithm for finding a particular value in an array, that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

Q 2. Write code to do Linear Search in an Array?

Ans. /* Program for linear search an element */

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

void main()
{
      int array[10];
      int i, N, keynum, found=0;
     clrscr();

    printf("Enter the value of N\n");
    scanf("%d",&N);

   printf("Enter the elements one by one\n");
   for(i=0; i<N ; i++)
       scanf("%d",&array[i]);

   printf("Input array is\n");
   for(i=0; i<N ; i++)
     printf("%d\n",array[i]);

  printf("Enter the element to be searched\n");
 scanf("%d", &keynum);

 /* Linear search begins */
i=0;
While(i < n) && (found==0)
 {
     if( keynum == array[i] )
           found = 1;
```

```
i++;
  }
if ( found == 1)
printf("SUCCESSFUL SEARCH\n");
 else
printf("Search is FAILED\n");

} /* End of main */
```

Q 3. How to do Binary Search in an Array?

Ans. A Binary Search or Half-interval search algorithm finds the position of a specified value (the input "key") within a sorted array. In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match, then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special "Not found" indication is returned.

Q 4. Write 'C' code to do Binary Search in an Array?

Ans.

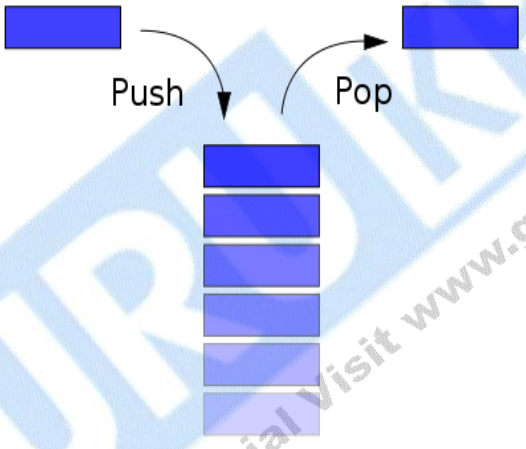
```
#include <stdio.h>
#include <conio.h>

void main()
{
        int array[10];
        int i, n, elem, found=0;
       clrscr();

      printf("Enter the value of N\n");
      scanf("%d",&n);

     printf("Enter the elements one by one\n");
     for(i=0; i<n ; i++)
         scanf("%d",&array[i]);

     printf("Input array is\n");
```

```
    for(i=0; i<n; i++)
       printf("%d\n",array[i]);

    printf("Enter the element to be searched\n");
   scanf("%d", &elem);

    /* Binary search begins */
  l=0;
            u=n-1;
            While((l<u) && (found==0))
            {
                        mid=(l+u)/2;
                        if(a[mid]==elem)
                        found=1;
            else
                        if(a[mid]>elem)
                                    u=mid-1;
                        else
                                    l=mid+1;
            }
            mid=mid+1;
  if ( found == 1)
                        printf("SUCCESSFUL SEARCH\n");
  else
                        printf("Search is FAILED\n");

} /* End of main */
```

STACK

Q 1. What is Stack?

Ans. A **stack** is a last in, first out (LIFO) abstract data type and linear data structure. A stack can have any abstract data type as an element, but is characterized by two fundamental operations, called *push* and *pop*. The push operation adds a new item to the top of the stack, or initializes the stack if it is empty. If the stack is full and does not contain enough space to accept the given item, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack. A pop either reveals previously concealed items, or results in an empty stack, but if the stack is empty then it goes into underflow state (It means no items are present in stack to be removed).



Q 2. Write a Program for Stack implementation through Array?

Ans.

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int stack[MAXSIZE];
int top;     //index pointing to the top of stack
void main()
{
        void push(int);
        int pop();
        int will=1,i,num;
        while(will ==1)
```

```
                {
                        printf("MAIN MENU:
                1.Add element to stack
                2.Delete element from the stack
  ");
  Scanf ("%d", &will);

  switch(will)
  {
                case 1: printf("Enter the data... ");
                        scanf("%d", &num);
                         push(num);
                        break;
                 case 2: num=pop();
                        printf("Value returned from pop function is  %d ",num);
                        break;
                default:  printf ("Invalid Choice . ");
  }
printf("Do you want to do more operations on Stack ( 1 for yes, any other key
to exit) ");
  scanf("%d" , &will);
      } //end of  outer while
}          //end of main
void push(int elem)
{
  if(isfull())
                {
                        printf("STACK FULL");
                        return;
                }
  top++;
  stack[top]=elem;
  }
}

int pop()
{
  int a;
  if(isEmpty())
  {
                printf("STACK EMPTY");
```

```
                return 0;
        }
        a=stack[top];
        top--;
        }
        return(a);
}

int isFull()
{
        if(top> MAXSIZE)
                return 0;
        else
                return 1;
}
int isEmpty()
{
        if(top<=0)
                return 0;
        else
                return 1;
}
```

Q 3. What are various Applications of Stack?

Ans. Some of the applications of stack are :

(i) Evaluating arithmetic expression
(ii) Balancing the symbols
(iii) Towers of Hanoi
(iv) Function Calls.
(v) 8 Queen Problem.

Q 4. How to Evaluating Arithmetic Expression?

Ans. There are 3 different ways of representing the algebraic expression. They are

* INFIX NOTATION
* POSTFIX NOTATION
* PREFIX NOTATION

INFIX NOTATION

In Infix notation, the arithmetic operator appears between the two operands to which it is being applied. For example: - A / B + C

POSTFIX NOTATION

The arithmetic operator appears directly after the two operands to which it applies. also called reverse polish notation. ((A/B) + C)For example: - AB / C +

PREFIX NOTATION

The arithmetic operator is placed before the two operands to which it applies. Also called as polish notation. ((A/B) + C)For example: - +/ABC

INFIX PREFIX (or) POLISH POSTFIX (or) REVERSEPOLISH

1. (A + B) / (C - D) /+AB - CD AB + CD - /
2. A + B*(C - D) +A*B - CD ABCD - * +
3. X * A / B - D - / * XABD X A*B/D-
4. X + Y * (A - B) / +X/*Y - AB - CD XYAB - *CD - / +(C - D)
5. A * B/C + D + / * ABCD AB * C / D +

To evaluate an arithmetic expressions, first convert the given infix expression to postfix expression and then evaluate the postfix expression using stack.

Infix to Postfix Conversion

Read the infix expression one character at a time until it encounters the delimiter. "#"

Step 1 : If the character is an operand, place it on to the output.

Step 2: If the character is an operator, push it onto the stack. If the stack operator has a higher or equal priority than input operator then pop that operator from the stack and place it onto the output.

Step 3: If the character is a left parenthesis, push it onto the stack.

Step 4: If the character is a right parenthesis, pop all the operators from the stack till item counters left parenthesis, discard both the parenthesis in the output.

Evaluating Postfix Expression

Read the postfix expression one character at a time until it encounters the delimiter `#'.

Step 1: - If the character is an operand, push its associated value onto the stack.

Step 2: - If the character is an operator, POP two values from the stack, apply the operator to them and push the result onto the stack.

Infix to Postfix Example

A + B * C - D / E

	Infix	Stack(bot->top)	Postfix
a)	A + B * C - D / E		
b)	+ B * C - D / E		A
c)	B * C - D / E	+	A
d)	* C - D / E	+	A B
e)	C - D / E	+ *	A B
f)	- D / E	+ *	A B C
g)	D / E	+ -	A B C *
h)	/ E	+ -	A B C * D
i)	E	+ - /	A B C * D
j)		+ - /	A B C * D E
k)			A B C * D E / - +

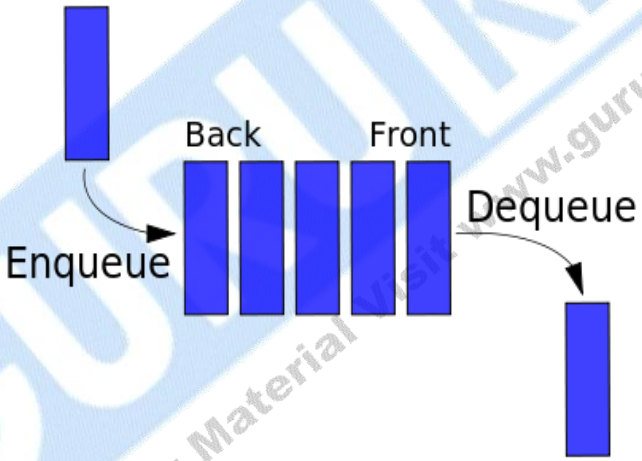
QUEUE

Q 1. What is Queue?

Ans. A **Queue** is a **first in, first out** (FIFO) abstract data type and linear data structure. A Queue can have any abstract data type as an element, but is characterized by two fundamental operations, called Enqueue and Dequeue. Queue has two pointers **Front** and **Rear.**

The **Enqueue** operation adds a new item to the front of the Queue, or initializes the queue if it is empty. If the Queue is full and does not contain enough space to accept the given item, the Queue is then considered to be in an overflow state.

The **Dequeue** operation removes an item from the front of the queue. A dequeue either reveals previously concealed items, or results in an empty queue, but if the queue is empty then it goes into underflow state (It means no items are present in queue to be removed).



While dealing with the circular queue we will use the following assumptions:

1. Front will always be pointing to the first element of the queue.
2. Each time a new element is inserted into the queue, the value of rear is incremented by one i.e., Rear = (Rear + 1);
3. Each time when an existing element is deleted from the queue, the value of rear is incremented by one i.e., Front = (Front + 1);
4. Front is the initial or first location of the queue where Rear indicates the last entry location in the queue.

Q 2. Write a code in 'C' for Linear Queue through Array?

Ans.

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int queue[MAXSIZE];
int front=-1, rear=-1;          //index pointing to the top of stack
void main()
{
        void enqueue(int);
        int dequeue();
        int will=1,i,num,ch;
        while(will ==1)
        {
                printf("MAIN MENU:
                1. Enqueue
                2.Dequeue
                3. Check Queue Full
                4. Check Queue Empty ");

                Scanf ("%d", &ch);

                switch(ch)
                {
                 case 1: printf("Enter the data to add... ");
                         scanf("%d", &num);
                         enqueue(num);
                         break;
                  case 2: num=dequeue();
                         if(num==-1)
                                 printf("Queue is empty");\
                         else

                        printf("Value returned from pop function is  %d
                ",num);
                         break;
                   case 3 : isFull();
                         break;
                   case 4 : isEmpty();
                         break;
```

```
                    default:  printf ("Invalid Choice . ");
  }
            printf("Do you want to do more operations on Stack ( 1 for yes, any
  other key to exit) ");
  scanf("%d" , &will);
      } //end of  outer while
}          //end of main
void enqueue (int elem)
{
  if(isfull())
            {
                    printf("QUEUE FULL");
                    return;
            }
  else{
            if(front==-1)
                    front=rear=0;
            else
                    rear++;
            queue[rear]=elem;
  }
}
int dequeue()
{
  int elem;
  if(isEmpty())
  {
            return -1;
  }
  else
  {
            elem=queue[front];
            if (front==rear)
                    front=rear=-1;
            else
                    front++;
  }
  return(elem);
}
int isFull()
{
```

```
if(rear==MAXSIZE-1)
        return 0;
else
        return 1;
}
int isEmpty()
{
if((front==-1) && (front==rear))
        return 0;
else
        return 1;
}
```

Q 3. What is a Circular Queue?

Ans. A **circular queue** is one in which the insertion of new element is done at the very first location of the queue if the last location of the queue is full. Suppose if we have a Queue of n elements then after adding the element at the last index i.e. (n-1)th , as queue is starting with 0 index, the next element will be inserted at the very first location of the queue which was not possible in the simple linear queue. That's why linear queue leads to wastage of the memory, and this flaw of linear queue is overcome by circular queue. So, in all we can say that the circular queue is a queue in which first element come right after the last element, that means a circular queue has a starting point but no end point.

Q 4. Write a code in 'C' for Circular Queue through Array?

Ans.

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int queue[MAXSIZE];
int front=-1, rear=-1;          //index pointing to the top of stack
void main()
{
        void enqueue(int);
        int dequeue();
        int will=1,i,num,ch;
        while(will ==1)
        {
        printf("MAIN MENU:
```

```
                1. Enqueue
                2.Dequeue
                3. Check Queue Full
                4. Check Queue Empty ");

                Scanf ("%d", &ch);

                switch(ch)
                {
                  case 1: printf("Enter the data to add... ");
                          scanf("%d", &num);
                          enqueue(num);
                          break;
                   case 2: num=dequeue();
                          if(num= = -1)
                                  printf("Queue is empty");\
                          else
                                  printf("Value returned from pop function is  %d
                          ",num);
                          break;
                    case 3 : isFull();
                          break;
                    case 4 : isEmpty();
                          break;
                   default:  printf ("Invalid Choice . ");
   }
                printf("Do you want to do more operations on Stack ( 1 for yes, any
   other key to exit) ");
   scanf("%d" , &will);
        } //end of  outer while
}          //end of main

void enqueue (int elem)
{
                if(isfull())
                {
                          printf("QUEUE FULL");
                          return;
                }
                else{
                          if(rear==-1)
```

```
                    front=rear=0;
                else
                    rear=(rear+1)% MAXSIZE;
                queue[rear]=elem;
        }
}
int dequeue()
{
        int elem;
        if(isEmpty())
        {
                return -1;
        }
        else
        {
                elem=queue[front];
                if (front==rear)
                        front=rear=-1;
                else
                        front= (front+1) % MAXSIZE ;
        }
        return(elem);
}
int isFull()
{
        If(((front==0) && (rear==MAXSIZE-1)) || (front=rear+1))
                return 0;
        else
                return 1;
}
int isEmpty()
{
        if((front==-1) && (front==rear))
                return 0;
        else
                return 1;
}
```

1. Exit.

SORTING

Q 1. What is selection sort?

Ans. This is the simplest sorting method. In order to sort data in ascending order, the 1st element is compared to all other elements and if found greater than the compared element, then they are interchanged. So after the first iteration the smallest element is placed at the 1st position. The same procedure is repeated for the 2nd element and so on for all the elements.

Example: Selection Sort

[0]	[1]	[2]	[3]	[4]	
5	1	3	7	2	find min
1	5	3	7	2	swap to index 0
1	5	3	7	2	find min
1	2	3	7	5	swap to index 1
1	2	3	7	5	find min
1	2	3	7	5	swap to index 2
1	2	3	7	5	find min
1	2	3	5	7	swap to index 3

Q 2. Write a 'C' code for selection sort?

Ans.

```
#include<stdio.h>
main ()
{
int array[100], size, i, j, min, swap;

printf("Enter number of elements\n");
scanf("%d", &size);

printf("Enter %d integers\n", size);
for ( i = 0 ; i < size ; i++ )
        scanf("%d", &array[i]);

for ( i = 0 ; i < ( size - 1 ) ; i++ )
{
        min = i;
        for ( j = i + 1 ; j < size ; j++ )
        {
```

```
                    if ( array[min] > array[j] )
                    min = j;
            }
            if ( min != i )
            {
                    swap = array[i];
                    array[i] = array[min];
                    array[min] = swap;
            }
}
  printf("Sorted list in ascending order:\n");
  for ( i = 0 ; i < size ; i++ )
            printf("%d\n", array[i]);

  return 0;
  }
```

Q 3. What is bubble sort?

Ans. **Bubble sort**, often incorrectly referred to as **sinking sort**, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a **comparison sort.**

For Example

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort algorithm. In each step, elements written in **bold** are being compared. Three passes will be required.

First Pass:

(**5 1** 4 2 8) ⟶ (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps them.

(1 **5 4** 2 8) ⟶ (1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) ⟶ (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) ⟶ (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) ⟶ (**1 4** 2 5 8)

(1 **4** **2** 5 8) ⟶ (1 **2** **4** 5 8), Swap since 4 > 2
(1 2 **4** **5** 8) ⟶ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) ⟶ (1 2 4 **5** **8**)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:
(**1** **2** 4 5 8) ⟶ (**1** **2** 4 5 8)
(1 **2** **4** 5 8) ⟶ (1 **2** **4** 5 8)
(1 2 **4** **5** 8) ⟶ (1 2 **4** **5** 8)
(1 2 4 **5** **8**) ⟶ (1 2 4 **5** **8**)

Q 4. What is Insertion sort?

Ans. Insertion sort is an elementary sorting algorithm. Here, sorting takes place by inserting a particular element at the appropriate position, that's why the name- insertion sorting.

- In the First iteration, second element A[1] is compared with the first element A[0].
- In the second iteration third element is compared with first and second element.
- In general, in every iteration an element is compared with all the elements before it.
- While comparing if it is found that the element can be inserted at a suitable position, then space is created for it by shifting the other elements one position up and inserts the desired element at the suitable position. This procedure is repeated for all the elements in the list.

```
#include<stdio.h>
int main()
{
 int i,j,size,temp,a[20];
 clrscr();
 printf("\nEnter size of the array: ");
 scanf("%d",&size);
 printf("\nEnter elements in to the array:");
 for(i=0;i<size;i++)
    scanf("%d",&a[i]);
 for(i=1;i<size;i++)
 {
```

```
        temp=a[i];
        j=i-1;
        while((j>=0)&&(temp<a[j]))
        {
                a[j+1]=a[j];
               j=j-1;
        }
        a[j+1]=temp;
 }
 printf("\nAfter sorting the elements are: ");
 for(i=0;i<size;i++)
    printf(" %d",a[i]);
 return 0;
}
```

Q 5. What is Merge sort?

Ans. Conceptually, merge sort works as follows:

1. Divide the unsorted list into two sublists of about half the size
2. Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
3. Merge the two sorted sublists back into one sorted list.

For Example

```
#include<stdio.h>
int main()
{
        int i,j,size,,a[50];
        clrscr();
        printf("\nEnter size of the array: ");
        scanf("%d",&size);
        printf("\nEnter elements in to the array:");
        for(i=0;i<size;i++)
                scanf("%d",&a[i]);
        mergesort(a,0,size-1)
        printf("\nAfter sorting the elements are: ");
        for(i=0;i<size;i++)
                printf(" %d",a[i]);

}
```

```
void mergesort(int a[], int low, int high)
{
int mid;
if(low<high)
{
        mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,high,mid);
}
}
merge(int a[], int low, int high, int mid)
{
 int i, j, k, c[50];
i=low;
j=mid+1;
k=low;
while((i<=mid)&&(j<=high))
{
      if(a[i]<a[j])
     {
             c[k]=a[i];
              k++;
              i++;
     }
    else
    {
           c[k]=a[j];
            k++;
            j++;
    }
}
while(i<=mid)
{
         c[k]=a[i];
          k++;
          i++;
}
while(j<=high)
{
       c[k]=a[j];
```

```
            k++;
            j++;
        }
        for(i=low;i<k;i++)
        {
            a[i]=c[i];
        }
    }
```

Q 6. Write a C program to implement Radix Sort

Ans.

```
#include <stdio.h>
#define MAX 5
#define SHOWPASS
void print(int *a, int n)
{
  int i;
  for (i = 0; i < n; i++)
    printf("%d\t", a[i]);
}

void radixsort(int *a, int n)
{
  int i, b[MAX], m = 0, exp = 1;
  for (i = 0; i < n; i++)
  {
    if (a[i] > m)
      m = a[i];
  }

  while (m / exp > 0)
  {
    int bucket[10] =
    {
      0
    };
    for (i = 0; i < n; i++)
      bucket[a[i] / exp % 10]++;
    for (i = 1; i < 10; i++)
```

```
    bucket[i] += bucket[i - 1];
   for (i = n - 1; i >= 0; i--)
    b[--bucket[a[i] / exp % 10]] = a[i];
   for (i = 0; i < n; i++)
    a[i] = b[i];
   exp *= 10;

   #ifdef SHOWPASS
    printf("\nPASS   : ");
    print(a, n);
   #endif
  }
}


int main()
{
  int arr[MAX];
  int i, n;

  printf("Enter total elements (n < %d) : ", MAX);
  scanf("%d", &n);

  printf("Enter %d Elements : ", n);
  for (i = 0; i < n; i++)
   scanf("%d", &arr[i]);


  printf("\nARRAY  : ");
  print(&arr[0], n);

  radixsort(&arr[0], n);

  printf("\nSORTED : ");
  print(&arr[0], n);
  printf("\n");

  return 0;
}
```

Q 7. Write a 'C' code for Quick Sort ?

Ans.

```
#include<stdio.h>
void print(int a[])
{
        int i;
       for( i=0;i<=8;i++)
           printf("%d ",a[i]);
}
int quickSort(int a[], int low, int high)
{
          int  i = low ;
          int  j = high ;
          int temp=0;
          int pivot = a[(left+right)/2];

do
          {
                    while (a[i] < pivot)
                              i++;
                    while (a[j] > pivot)
                              j--;
                    if (i <= j)
                    {
                              temp = a[i];
                              a[i] = a[j];
                              a[j] = temp;
                              i++;
                              j--;
                    }
          }while (i <= j);

          if (low < j)
                    quickSort (a, low, j);
          if (i < high)
                    quickSort (a, i , high);

}
void main()
{
          int array[]={12,99,4,99,12,12,13,10,13};
          printf("Before sort:\n\n");
```

```
        print(array);
        quickSort(array,0,8);
        printf("\n\nAfter sort:\n\n");
        print(array);
        printf("");
}
```

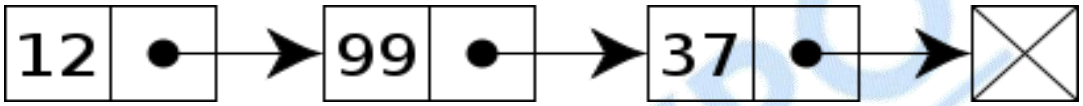
LINKED LIST

Q 1. What is linked list?

Ans. A linked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers i.e., each node is divided into two parts :

1. Information of the element, and
2. The link field or next pointer field containing address of the next node in the list.

For Example :

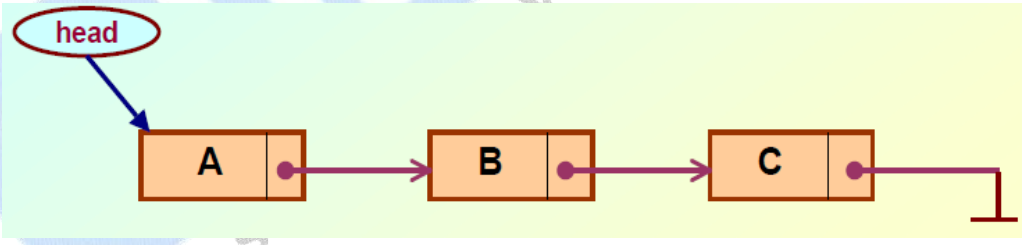


Q 2. How many types of Linked List are there?

Ans. Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

Linear singly-linked list (or simply linear list):

- Successive elements are connected by pointers
- Last element points to NULL.



Q 3. Write a 'C' code to create and traverse a linear linked list?

Ans. to create a linked list at first we need to design the structure of node i.e. what kind of data it is having. After creating node we will make a start node.

```
#include<conio.h>
#include<stdio.h>
void creation(int);
void display();
typedef struct linkedlist
```

```
{
 int num;
 struct linkedlist *next;
}node;

Node *head=NULL,*p=NULL;
void main()
{
 int n1,ch;
 clrscr();
 printf("enter the no of nodes to be entered : ");
 scanf("%d",&n1);
 do
 {
   clrscr();
   printf("\n1.creation\n2.insertion\n3.deletion\n4.display\n5.exit\n");
   printf("\nenter ur choice : ");
   scanf("%d",&ch);
   switch(ch)
   {
        case 1:
                creation(n1);
                break;
case 2:
                display();
                getch();
                break;
case 3:
                exit(0);
   }
 }while(ch!=3);
 getch();
}

/*  CREATION   */
void creation(int n1)
{
        int i, n ;
        head=((struct list *) malloc(sizeof(struct list)));
 p=head;
        for(i=0;i<n1;i++)
```

```
        {
                printf("enter the %d node : ",i+1);
                scanf("%d",&n);
                p->num = n;
                p->next = ((struct list *)malloc(sizeof(struct list)));
                p=p->next;
        }
        p->next=0;
}
/*  DISPLAY   */
void display()
{
        p=head;
         printf("\nthe nodes entered are : ");
         while(p->next>0)
        {
                printf ("%d\t", p->num);
                p=p->next;
        }
}
```

Q 4. How to insert a node at first position in linear linked list?

Ans: In dynamic addition of NODE, we assume that the list is already there, means of non-zero length and there is a NODE pointer "target" that points to a node after which the operation(addnode) has to be done.

Algorithm:-

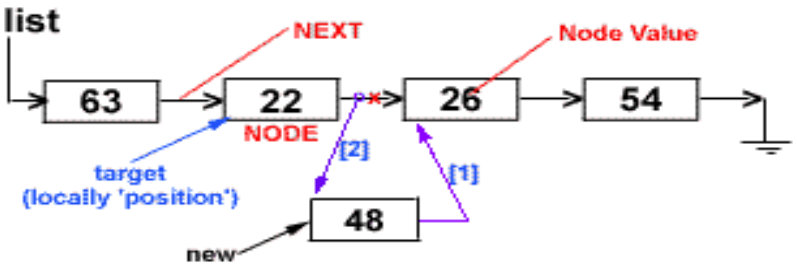
Step-1: Get the value for your NEW node to be added to the list and its Target position

Step-2: Create a NEW, empty node by calling malloc (). If malloc () returns no error then go to step-3 or else say "Memory shortage"

Step-3: Put the value inside the NEW node's node value field

Step-4: Add this NEW node at the desired position (pointed by the "target") in the LIST

Step-5: Go to step-1 till you have more values to be added to the LIST




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

typedef struct linklist
{
   int data;
   struct linklist *next;
}node;

void main()
{
        node* create(node*);
        void display(node*);
        node* insert_beg(node*,int);
        node* delete_first(node*);
        node *head=NULL;
        int x;
        clrscr();

        head=create(head);
        display(head);
        printf("\nenter element to insert");
        scanf("%d",&x);
        head=insert_beg(head,x);
        printf("\ncreated link list is :");
        display(head);
        getch();
}

node* create(node *head)
{
        node *insert(node*,int);
```

```
        int x;
        printf("\nenter element to be inserted");
        scanf("%d",&x);
        while(x!=99)
        {
                head=insert(head,x);
                printf("\nenter element");
                scanf("%d",&x);
        }
        return(head);
}
node* insert(node *head,int x)
{
        node* getnode(int);
        node *p,*q;
        p=getnode(x);
        q=head;
        if(head==NULL)
                head=p;
        else
        {
                while(q->next!=NULL)
                        q=q->next;
                q->next=p;
        }
        return(head);
}
node* insert_beg(node *head,int x)
{
        node* getnode(int x);
        node *p;
        p=getnode(x);
        if(head==NULL)
                head=p;
        else
        {
                p->next=head;
                head=p;
        }
        return(head);
}
```

```
node* getnode(int x)
{
        node *p;
        p=(node*)malloc(sizeof(node));
        p->data=x;
        p->next=NULL;
        return(p);
}
void display(node *head)
{
            node *p;
               printf("\nvalues of the entered list is:");
                p=head;
               while(p!=NULL)
               {
                        printf("%d",p->data);
                        p=p->next;
               }

        node *p;
        p=head;
        while(p!=NULL)
        {

               p=p->next;
        }
}
```

Q 5. How to delete a node from a first position in linear linked list?

Ans: Algorithm:-

Step-1: Take the value in the 'node value' field of the TARGET node in any intermediate variable

Step-2: Make the previous node of TARGET to point to where TARGET is pointing

Step-3: Free the TARGET

Step-4: Return the value in that intermediate variable

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

```
typedef struct linklist
{
   int data;
   struct linklist *next;
}node;

void main()
{
        node* create(node*);
        void display(node*);
        node* insert_beg(node*,int);
        node* delete_first(node*);
        node *head=NULL;
        int x;
        clrscr();

        head=create(head);
        display(head);
        head=delete_first(head);
        display(head);
        getch();
}

node* create(node *head)
{
        node *insert(node*,int);
        int x;
        printf("\nenter element to be inserted");
        scanf("%d",&x);
        while(x!=99)
        {
            head=insert(head,x);
            printf("\nenter element");
            scanf("%d",&x);
        }
        return(head);
}
node* insert(node *head,int x)
{
        node* getnode(int);
        node *p,*q;
```

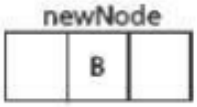
```
        p=getnode(x);
        q=head;
        if(head==NULL)
                head=p;
        else
        {
                while(q->next!=NULL)
                        q=q->next;
                q->next=p;
        }
        return(head);
}
node* delete_first(node *head)
{
        node *p;
        p=head;
        if(head!=NULL)
                head=head->next;
        free(p);
        return(head);
}
node* getnode(int x)
{
        node *p;
        p=(node*)malloc(sizeof(node));
        p->data=x;
        p->next=NULL;
        return(p);
}
void display(node *head)
{
        node *p;
        printf("\nvalues of the entered list is:");
         p=head;
        while(p!=NULL)
        {
                printf("%d",p->data);
                p=p->next;
        }
}
```

Q 6. What is Doubly LinkedList?

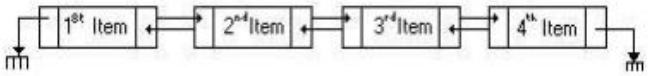
Ans: In doubly Linked list

A double linked list is defined as a collection of nodes in which each nodes has three parts:

- Data : It contains the data value for the node,
- Leftlink : leftlink contains the address of node before it
- Rightlink : rightlink contains the address of node after it.



This is how a doubly linked list looks like:



Note that

- Each node point to previous node and next node.
- Since there is no node before first node, so first node's llink contains null.
- Similarly, for the last node. There is no node after last node, hence last node's rlink contains null.

Here is the data type of node for Double Linked List

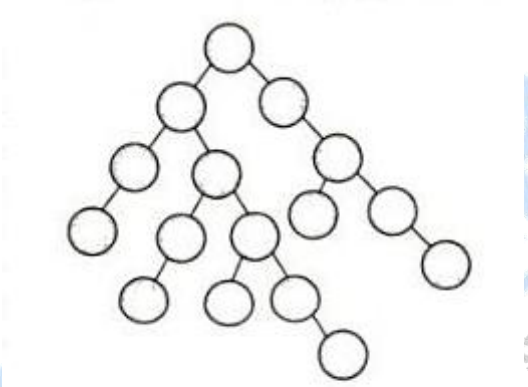
```
typedef stuct node
{
int data;
node *llink;
node *rlink;
}
```

TREES

Q 1. What is Tree Data Structure?

Ans:

- A tree is called non-linear data structure because the elements of this data structure are arranged in a non-linear fashion i.e., they use two dimensional representation.
- The tree data structure is most suitable where the hierarchy relationship among data are to be maintained.
- Advantage of this data structure is that insertion, deletion, searching etc., are more efficient than linear data structures



Tree—a non-linear (two-dimensional) representation of data

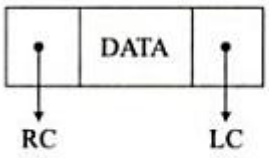
Definition

A tree is a collection of elements called nodes, one of which is designated as root along with a relation that places a hierarchical structure on the nodes.

Q 2. Explain various terminologies used in tree?

Ans: Node

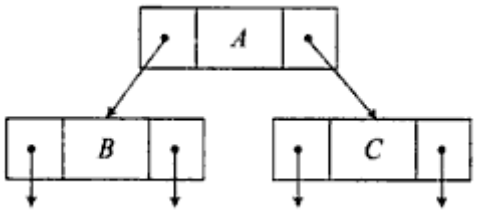
- This is the main component of any tree structure
- The concept of node in the tree is same as used in linked list
- Each individual element of a tree can be called a node.
- The node of the tree stores the actual data along with the links to other nodes.



Structure of a node in a tree

Parent

- The parent of a node is the immediate predecessor of that node.
- **A** is the parent nodes **B** and **C**



Child

- The immediate successors of a node are called child nodes.
- In fig ., **B** and **C** are the child nodes of **A.**
- A child which is placed at the left side is called the left child
- A child which is placed at the right side is called the right child

Link (Branch or Edge)

- A link is a pointer to a node in the tree
- A node can have more than one link
- In Fig., the node **A** has two links

Root

- A root is a specially designated node in a tree
- It is a node which has no parent.
- In fig., the node A is the root of the tree
- There can be only one root node in a tree

Leaf Node(or External node)

- A leaf node is a node which does not have any child node.
- In Fig., nodes B and C are leaf nodes

Level

- The level of a node in the tree is the rank of the hierarechy.

- The root node is placed in level 0.
- If a node is at level L the its child is at level L+1 and its parent is at level L-1

Q 3. Explain the different definitions for Rooted Tree.

Ans. Definitions for Rooted Trees :

- **A directed edge** refers to the link from the parent to the child (the arrows in the picture of the tree).
- **The root node** of a tree is the node with no parents. There is at most one root node in a rooted tree.
- **A leaf** is a node that has no children.
- **The depth of a node** n is the length of the path from the root to the node. The set of all nodes at a given depth is sometimes called a level of the tree. The root node is at depth zero.
- **The height of a tree** is the length of the path from the root node to its furthest leaf. A tree with only a root node has a height of zero.
- **Siblings** are nodes that share the same parent node.
- If a **path** exists from node p to node q, where node p is closer to the root node than q, then p is an ancestor of q and q is a descendant of p.
- The **size of a node** is the number of descendants it has including itself.

Q 4. Explain about the different types of Binary Tree.

Ans. Types of Binary Trees*:*

- A **rooted binary tree** is a rooted tree in which every node has at most two children.
- A **full binary tree**, or proper binary tree, is a tree in which every node has zero or two children.
- A **perfect binary tree** (sometimes complete binary tree) is a full binary tree in which all leaves are at the same depth.
- A **complete binary tree** may also be defined as a full binary tree in which all leaves are at depth n or n-1 for some n. In order for a tree to be the latter kind of complete binary tree, all the children on the last level must occupy the leftmost spots consecutively, with no spot left unoccupied in between any two. For example, if two nodes on the bottommost level each

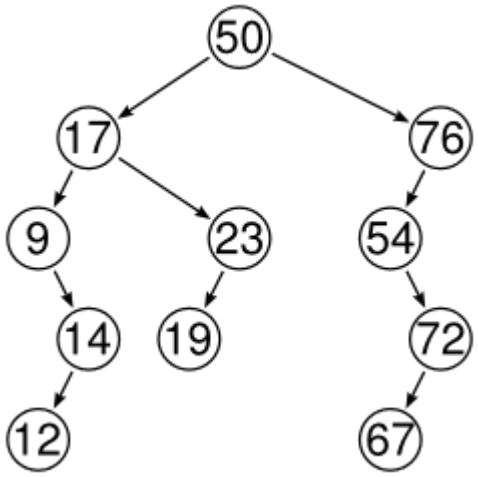
occupy a spot with an empty spot between the two of them, but the rest of the children nodes are tightly wedged together with no spots in between, then the tree cannot be a complete binary tree due to the empty spot.

- A **rooted complete binary** tree can be identified with a free magma.
- An **almost complete binary tree** is a tree in which each node that has a right child also has a left child. Having a left child does not require a node to have a right child. Stated alternately, an almost complete binary tree is a tree where for a right child, there is always a left child, but for a left child there may not be a right child.
- The number of nodes n in a complete binary tree can be found using this formula: n = 2h + 1 - 1 where h is the height of the tree.
- The number of leaf nodes n in a complete binary tree can be found using this formula: n = 2h where h is the height of the tree.

Q.12. Define AVL Trees.

Ans.: An example of an unbalanced non-AVL tree

In computer science, an AVL tree is a self-balancing binary search tree, and the first such data structure to be invented[citation needed]. In an AVL tree the heights of the two child subtrees of any node differ by at most one, therefore it is also called height-balanced. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases. Additions and deletions may require the tree to be rebalanced by one or more tree rotations.



The AVL tree is named after its two inventors, G.M. Adelson-Velsky and E.M. Landis, who published it in their 1962 paper "An algorithm for the organization of information."

The balance factor of a node is the height of its right subtree minus the height of its left subtree. A node with balance factor 1, 0, or -1 is considered balanced. A node with any other balance factor is considered unbalanced and requires rebalancing the tree. The balance factor is either stored directly at each node or computed from the heights of the subtrees.

M.C.Q's Questions

Q 1. The memory address of the first element of an array is called

a. floor address
b. foundation address
c. first address
d. base address

Q 2. The memory address of fifth element of an array can be calculated by the formula

a. LOC(Array[5]=Base(Array)+w(5-lower bound), where w is the number of words per memory cell for the array
b. LOC(Array[5])=Base(Array[5])+(5-lower bound), where w is the number of words per memory cell for the array
c. LOC(Array[5])=Base(Array[4])+(5-Upper bound), where w is the number of words per memory cell for the array
d. None of above

Q 3. Which of the following data structures are indexed structures?

a. linear arrays
b. linked lists
c. both of above
d. none of above

Q 4. Which of the following is not the required condition for binary search algorithm?

a. The list must be sorted
b. there should be the direct access to the middle element in any sublist
c. There must be mechanism to delete and/or insert elements in list
d. none of above

Q 5. Which of the following is not a limitation of binary search algorithm?

a. must use a sorted array
b. requirement of sorted array is expensive when a lot of insertion and deletions are needed

c. there must be a mechanism to access middle element directly
d. binary search algorithm is not efficient when the data elements are more than 1000.

Q 6. Two dimensional arrays are also called

a. tables arrays
b. matrix arrays
c. both of above
d. none of above

Q 7. A variable P is called pointer if

a. P contains the address of an element in DATA.
b. P points to the address of first element in DATA
c. P can store only memory addresses
d. P contain the DATA and the address of DATA

Q 8. Which of the following data structure can't store the non-homogeneous data elements?

a. Arrays
b. Records
c. Pointers
d. None

Q 9. Which of the following data structure store the homogeneous data elements?

a. Arrays
b. Records
c. Pointers
d. None

Q 10. Each data item in a record may be a group item composed of sub-items; those items which are indecomposable are called

a. elementary items
b. atoms
c. scalars
d. all of above

Answer

1. The memory address of the first element of an array is called
d. base address

---

2. The memory address of fifth element of an array can be calculated by the formula
a. LOC(Array[5]=Base(Array)+w(5-lower bound), where w is the number of words per memory cell for the array

---

3. Which of the following data structures are indexed structures?
a. linear arrays

---

4. Which of the following is not the required condition for binary search algorithm?
c. There must be mechanism to delete and/or insert elements in list

---

5. Which of the following is not a limitation of binary search algorithm?
d. binary search algorithm is not efficient when the data elements are more than 1000.

---

6. Two dimensional arrays are also called
c. both of above

---

7. A variable P is called pointer if
a. P contains the address of an element in DATA.

---

8. Which of the following data structure can't store the non-homogeneous data elements?
a. Arrays

---

9. Which of the following data structure store the non-homogeneous data elements?

b. Records

---

10. Each data item in a record may be a group item composed of sub-items; those items which are indecomposable are called
d. all of above

Keywords

nil

nil is a possible value of any data type in Clojure. nil has the same value as Java null. The Clojure conditional system is based around nil and false, with nil and false representing the values of logical falsity in conditional tests - anything else is logical truth. In addition, nil is used as the end-of-sequence sentinel value in the sequence protocol.

Keywords

Keywords are symbolic identifiers that evaluate to themselves. They provide very fast equality tests. Like Symbols, they have names and optional namespaces, both of which are strings. The leading ':' is not part of the namespace or name.

Keywords implement IFn for invoke() of one argument (a map) with an optional second argument (a default value). For example (:mykey my-hash-map :none) means the same as (get my-hash-map :mykey :none).

Symbols

Symbols are identifiers that are normally used to refer to something else. They can be used in program forms to refer to function parameters, let bindings, class names and global vars. They have names and optional namespaces, both of which are strings. Symbols can have metadata.

Collections

All of the Clojure collections are immutable and persistent. In particular, the Clojure collections support efficient creation of 'modified' versions, by utilizing structural sharing, and make all of their performance bound guarantees for persistent use. The collections are efficient and inherently thread-safe. Collections are represented by abstractions, and there may be one or more concrete realizations. In particular, since 'modification' operations yield new collections, the new collection might not have the same concrete type as the source collection, but will have the same logical (interface) type.

ArrayMaps

When doing code form manipulation it is often desirable to have a map which maintains key order. An array map is such a map - it is simply implemented as an array of key val key val... As such, it has linear lookup performance, and is only suitable for *very small* maps. It implements the full map interface. New ArrayMaps

can be created with the array-map function. Note that an array map will only maintain sort order when un-'modified'. Subsequent assoc-ing will eventually cause it to 'become' a hash-map.

Lists

Lists are collections.

Sets

Sets are collections of unique values.

There is literal support for hash-sets:

```
#{:a :b :c :d}
-> #{:d :a :b :c}
```