Yashwantrao Chavan
Maharashtra
Open University

[Established by Government of Maharashtra]

ज्ञानगंगा घरोघरी

Student Handout Book

CMP 215

Data Structures through C++

Yashwantrao Chavan Maharashtra Open University

Vice-Chancellor : Dr. Rajan Welukar

SCHOOL OF COMPUTER SCIENCE : SCHOOL COUNCIL

COURSE DEVELOPED BY

Kanetkar's ICIT Pvt. Ltd.
44-A, Hill Road, Gokul Peth
Nagapur

PRODUCTION

Shri. Anand Yadav
Manager
Print Production Centre YCMOU, Nashik

This Book is developed under DEC Development Grant

- **First Published by :** YCMOU, Nashik (September 2009)
- **Publication No. :** 1817
- **Cover Design :** Avinash Bharne
- **Printed by :** Shri. Sanket Pathak, M/s Amal Offset Pvt. Ltd. Satpur, Nashik-7
- **Published by :** Shri. Prakash Wani, Registrar, Y. C. M. Open University, Nashik - 422 222.

CMP215

[Data Structures through C++ : Student Handout Book]

Introduction To Data Structures

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- What is the meaning of Data Structure
- Why are they important
- What different types exist
- How do you select the correct one
- Does the selection matter

What Are Data Structures

- Proper organization leads to efficient access
- Examples:
  - Notes in a wallet
  - Certificates in a file
  - Chapters in a book
  - Words in a dictionary
- Same is true about storing data in computer
- If stored properly, it can be used efficiently
- So , DS = Storage with motive of efficient use

Should I Really Care

- I keep getting some or the other Bug in the S/W
- Slow working of a particular site
- Asking same data to be entered repeatedly
- Google search versus others
- It takes an eternity to sort data
- Moral...

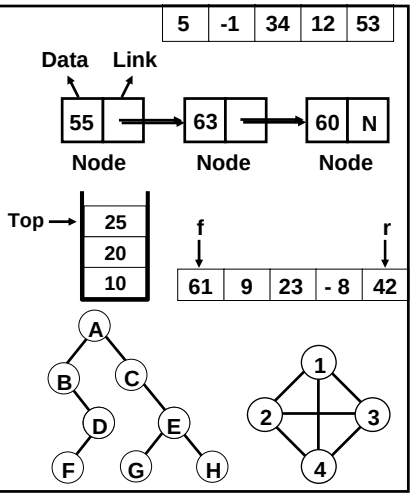
Good programming begins with right choice of DS

Properties Of A Good Data Structure

- Permits a variety of critical opns to be performed
- Uses less execution time
- Uses less memory space

Common DS

- Arrays
- Sparse Matrices
- Linked Lists
- Stacks
- Queues
- Trees
- Graphs



Which One To Use

- Time complexity
- Space complexity
- At times, ease of use
- At times, operations that we wish to perform
- At times, programming language used
- Some are suited better for some applications
  - Ex. B-Trees for Databases
  - Ex. Stacks for function calls

Impact Of Good Data Structures

- Ease of implementation
- Quality of implementation
- Quality of performance

Choices Available

- Implement Data Structures yourselves
- Use standard libraries:
  - C++'s Standard Template Library
  - Java's Collections Framework
  - Microsoft's .NET Framework

Searching

Asang Dani

Objectives

- Operations performed on an array
- What are algorithms
- Important features of algorithms
- Conventions to follow while writing an algorithm
- Linear search and Binary search

Algorithm

- Method of accomplishing a task in a finite number of steps
- Origin - Persian Mathematician - Abu Jaffer Al-Khowarizmi
- Aka - Recipe, Method, Technique, Procedure, Routine
- Important Features:
  - Input - Must have zero or more inputs
  - Output - Must have one or more outputs
  - Finiteness - Must terminate after finite no. of steps
  - Definiteness - Each step must be unambiguously defined
  - Effectiveness - All operations must be sufficiently basic
- Types:
  - Iterative - Repetition using loop
  - Recursive - Divide and Conquer

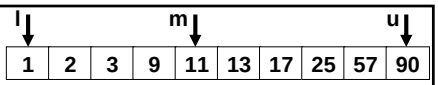
Array Operations

Operations	Description
Traversal	Processing each element in the array
Search	Finding the location of an element with a given value
Insertion	Adding a new element to an array
Deletion	Removing an element from an array
Sorting	Organizing the elements in some order
Merging	Combing two arrays into a single array

Linear Search

```
main( )
{
    int a[ ] = { 11, 2, 9, 13, 57, 25, 17, 1, 90, 3 } ;
    int i, num ;
    printf ( "Enter no. to search: " ) ;
    scanf ( "%d", &num ) ;
    for ( i = 0 ; i <= 9 ; i++ )
    {
        if ( a[ i ] == num )
            break ;
    }
    if ( i == 10 )
        printf ( "No such number in array" ) ;
    else
        printf ( "Number is at position %d", i ) ;
}
```

Binary Search




```
main( )
{
    int a[ ] = { 1, 2, 3, 9, 11, 13, 17, 25, 57, 90 } ;
    int l = 0, u = 9, m, num ;
    printf ( "Enter number to search: " ) ;
    scanf ( "%d", &num ) ;
    while ( l <= u )
    {
        m = ( l + u ) / 2 ;
        if ( a[m] == num )
        {
            printf ( "No. is at position %d ", m ) ;  exit( ) ;
        }
        a[m] > num ? ( u = m - 1 ) : ( l = m + 1 ) ;
    }
    printf ( "Element is not present in the array." ) ;
}
```

57

Searching & Frequency Count

Asang Dani
asang@ksetindia.com

Objectives

- Analyzing an algorithm
- Rate of increase for growth rate with respect to Big Oh notation

Analysis Of Algorithms

Determining

- Time Requirement
  - Difficulty
    - ◆ Machine dependent
    - ◆ Language dependent
    - ◆ Compiler dependent
  - Solution
    - ◆ Don't calculate exact time
    - ◆ Calculate approx. no. of operns. for given input
    - ◆ No. of opns α Time
  - ◆ Iterative - WD in loop
  - ◆ Recursive - WD in break & make
- Space Requirement
  - Cheaper - Not so relevant

Example - Biggest of 4

3 comparisons

Algorithm 1

```
if ( a > b )
   if ( a > c )
      if ( a > d )
         return a
      else
         return d
      end if
   else
      if ( c > d )
         return c
      else
         return d
      end if
   end if
else
   if ( b > c )
      if ( b > d )
         return b
      else
         return d
      end if
   else
      if ( c > d )
         return c
      else
         return d
      end if
   end if
end if
```

Algorithm 2

```
big = a
if ( b > big )
   big = b
end if
if ( c > big )
   big = c
end if
if ( d > big )
   big = d
end if
return big
```

What To Count

- Multiplication of Matrices - Number of multiplications
- Searching - Number of comparisons
- Sorting - Number of comparisons
- Count chars in a file

```
for ( i = 0 ; i <= 255 ; i + + )
   a[ i ] = 0 ;
while ( ( ch = getc ( fp ) ) != EOF )
   a[ ch ] + + ;
```

N	Opns. in Loop1	Opns. in Loop2 ✓	Total
500	256 + 256 + 256 (33%)	500 + 500 + 500 (66%)	2268
50000	256 + 256 + 256 (1%)	50000 + 50000 + 50000 (99%)	150768

Fibonacci Series

```
1    fibonacci ( )
2    {
3       old = 1 ;
4       new = 1 ;
5       n = 20 ;
6       for ( i = 1 ; i < n ; i + + )
7       {
8          a = old + new ;
9          printf ( "%d ", a ) ;
10         old = new ;
11         new = a ;
12      }
13   }
```

Line no.	No. of execution
3	1
4	1
5	1
6	n - 1
8	n - 1
9	n - 1
10	n - 1
11	n - 1
Total	5n - 2

Ignoring the const 5 & 2
complexity -> O(n)

Exercise

Determine the frequency counts for all statements in the following two program segments:

```
1    for ( i = 0 ; i < n ; i + + )
2    {
3       for ( j = 0 ; j < i ; j + + )
4       {
5          for ( k = 0 ; k < j ; k + + )
6          {
7             x + + ;
8          }
9       }
10   }
```

Line no.	No. of exec.
1	n
3	n * i
5	n * i * j
7	n * i * j

n (1 + i + 2 i j)
O (n)

```
1  i = 0
2  while ( i < n )
3  {
4     x = x + 1 ;
5      i =  i + 1 ;
6  }
```

O (n)

Line no.	No. of exec.
1	1
2	n
4	n
5	n

Rate of Increase

n	log n	n log n	n^2	n^3	2^n
1	0.0	0.0	1.0	1.0	2.0
2	1.0	2.0	4.0	8.0	4.0
5	2.3	11.6	25.0	125.0	32.0
10	3.3	33.2	100.0	1000.0	1024.0
15	3.9	58.6	225.0	3375.0	32768.0
20	4.3	86.4	400.0	8000.0	1048576.0
30	4.9	147.2	900.0	27000.0	1073741824.0
40	5.3	212.9	1600.0	64000.0	1099511627776.0
50	5.6	282.2	2500.0	125000.0	1125899906842620.0

O(1) - const., O(n) - linear, O(n^2) - quadratic, O(n^3) - cubic
O(2^n) - exponential

O(log n) is faster than O(n)
O(n log n) is faster than O(n^2) but not as good as O(n)

Exercise

For which range of values would the algorithm whose order of magnitude is n^3 be better than an algorithm whose order of magnitude is 2^n

n	n^3	2^n
1	1.0	2.0
2	8.0	4.0
5	125.0	32.0
6	216.0	64.0
7	343.0	128.0
8	512.0	256.0
9	729.0	512.0
10	1000.0	1024.0
15	3375.0	32768.0
20	8000.0	1048576.0
30	27000.0	1073741824.0
40	64000.0	1099511627776.0
50	125000.0	1125899906842620.0

Range	Better
n = 1	n^3
n = 2 to 9	2^n
n > 10	n^3

Analysis Of Searching Methods

Asang Dani

Objectives

- Tools to calculate time complexity
- Cases to be considered while analyzing algorithms
- Analysis of Linear search and Binary search

Classification Of Growth

- Rate of growth is dominated by largest term in an equation
- Neglect terms that grow more slowly
- Leftover is known as order of the algorithm
- Algos. are grouped into 3 categories based on their order
  - Big Omega - Ω (f) ✗

    If $g(x) \in \Omega(f)$, $g(n) >= cf(n)$ for all $n >= n_0$ (c = const.)

    Represents class of funcns that grow at least as fast as f
  - Big Oh - O(f) ✓

    If $g(x) \in O(f)$, $g(n) <= cf(n)$ for all $n >= n_0$ (c = const.)

    Represents class of funcns that grow no faster than f
  - Big Theta - θ(f)

    $\theta(f) = \Omega(f) \cap O(f)$ ✗

    Represents class of funcns that grow as fast as f

Analysis Of Linear Search

```
linearsearch ( int  *list, int  value, int  n )
{
   for ( i = 0 ; i < n ; i + + )
   {
      if ( value == list [ i ] )
         return  i ;
   }
   return  -1 ;
}
```

Best	Worst	Avg.
1	N	$\frac{N+2}{2}$

➡ Possible inputs
➡ Probability of each input

The value being searched is found in first location

➡ The value being searched matches the last elements in the list
➡ The value being searched is not present in the list

Average Case

$$A(N) = \left(\frac{1}{N+1}\right) * \left[\left(\sum_{i=1}^{N} i\right) + N\right]$$

Probability

$$A(N) = \left(\frac{1}{N+1}\sum_{i=1}^{N} i\right) + \left(\frac{1}{N+1} * N\right)$$

$$A(N) = \left(\frac{1}{\cancel{N+1}} * \frac{N(\cancel{N+1})}{2}\right) + \frac{N}{N+1}$$

$$A(N) = \frac{N}{2} + \frac{N}{N+1} = \frac{N}{2} + 1 - \frac{1}{N+1}$$

$$A(N) \approx \frac{N+2}{2}$$

(As n gets very large, $\frac{1}{N+1}$ becomes almost 0)

Analysis Of Binary Search

```
bisearch ( int  *a, int  x )
{
   lower = 0 ;  upper = 10 ;
   while ( lower <= upper )
   {
      mid = ( lower + upper ) / 2 ;
      switch ( compare ( x, a[ mid ] ) )
      {
         case  '>' :
           lower = mid + 1 ;  break ;
         case  '<' :
           upper = mid - 1 ;  break ;
         case  '=' :
           printf ( "%d ", mid ) ; exit( ) ;
      }
   }
}
```

Best	Worst
1	log (N + 1)

Halving nature of algo.

$N = 2^k - 1$
$2^{k-1} - 1, 1, 2^{k-1} - 1$

K = 3, $2^3 - 1 = 7$, N = 7
$2^{3-1} - 1, 1, 2^{3-1} - 1$
3, 1, 3

K = 2, $2^2 - 1 = 3$, N = 3
$2^{2-1} - 1, 1, 2^{2-1} - 1$
1, 1, 1

$N = 2^k - 1$
$\log_2 2^k = \log_2 (N + 1)$
$k \log_2 2 = \log_2 (N + 1)$
$k = \log_2 (N + 1)$

Hashing

Asang Dani

Objectives

- **Hashing Techniques**
  - **Division method**
  - **Mid – Square method**
  - **Folding method**
  - **Digit Analysis method**
  - **Linear and quadratic probing**

Hashing Functions

- **Division** ✓
- **Mid - Square**
- **Folding**
- **Digit Analysis**

Division

3229329 4231176 7621913 9812427 2178115 4031231

Store elements as per hash value
Hash value - index based
Hash value = no. % 10
If hash value clashes - collision
- chaining (not efficient)
- rehashing

1431327

0	
1	4031231
2	
3	7621913
4	
5	2178115
6	4231176
7	9812427
8	1431327
9	3229329

Linear Probing

Mid - Square

- Identifiers - A = 1, ..., Z = 26, 0 = 27, 1 = 28, ..., 9 = 36
- Find octal equivalent of each character
- Square the octal equivalent
- Use middle bits of square as hash value
- Table size = 2^r, r is no. of middle bits

X	X^1	$(X^1)^2$
A	01	1
B	02	4
...	...	...
Y	31	1701
Z	32	2000
0	33	2101
1	34	2204
A1	134	20420
A2	135	20711
CAT	030124	125620

Folding

- Distribute digits in multiple partitions
- Excluding last make all partitions equal
- Add partition values to get hash value

No. Of Digits	Partition
1	1
2	2
3	1, 2
4	1, 1, 2
5	2, 2, 1
6	1, 1, 1, 1, 2
7	3, 3, 1
8	3, 3, 2
9	2, 2, 2, 2, 1
...	...

CAT 030124

0	3	0	1	24	28

Digit Analysis

- Each identifier is interpreted as a no. using some radix r
- Same radix is used for other identifiers
- Digits in each identifier is examined
- Digits with skewed distribution are deleted
- Digits are deleted till balance digits are in range of HT

More Hashing

- Hash table contains buckets
- Each buckets may contain several slots
- Each slot can hold one record
- Collision - when two identifiers hash into same bucket
- Overflow - when new identifier is hashed into a full bucket
- If number of slots = 1 then Collision = Overflow
- Collision handling techniques:
  - Linear probing -
    search the bucket (f (x) + i) % b, for 0 <= i <= b - 1
  - Quadratic probing -
    search the bucket f (x)
    (f (x) + i^2) % b
    (f (x) - i^2) % b, for 1 <= i <= (b - 1) / 2

Linear Probing

4028 2133 1098 7915 6749 5141 3138

f (x) = no. % 10

key = (f(x) + i) % b

```
f ( int  no, int  *arr, int  b )
{
    int  i, j, initialpos ;
    j = no % 10 ;  intialpos = j ;
    for ( i = 1 ; arr[ j ]  !=  no && arr[ j ]  !=  0 ; i + + )
    {
        j = ( no % 10 + i ) % b ;
        if ( j  ==  initialpos )
        {
            printf ( "Array full" ) ;  return ;
        }
    }
    arr [ j ] = no ;
}
```

0	6749
1	5141
2	3138
3	2133
4	
5	7915
6	
7	
8	4028
9	1098

Quadratic Probing

4028	2133	1098	7915	6749	5141	3138

f (x) = no. % 10
key = f(x)
= f(x) + i^2) % b
= f(x) - i^2) % b
for 1 <= i <= (b -1) / 2

0	
1	5141
2	3138
3	2133
4	
5	7915
6	
7	6748
8	4028
9	1098

Sorting

Asang Dani

Objectives

- Selection Sort and its Analysis
- Bubble Sort and its Analysis
- Radix Sort

Selection Sort

```
main( )
{
   int  a[ ] = { 17, 6, 13,12, 2 } ;
   int  i,  j,  t ;
   for ( i = 0 ; i <= 3 ; i + + )
   {
     for ( j = i + 1 ; j <= 4 ; j + + )
     {
        if ( a[ i ] >a[ j ] )
        {
           t = a[ i ] ;  a[ i ] = a[ j ] ;
           a[ j ] = t ;
        }
     }
   }
   for ( i = 0 ; i <= 4 ; i + + )
      printf ( "%d", a[ i ] ) ;
}
```

17	6	13	12	2	i j
6	17	13	12	2	0 - 1
6	17	13	12	2	0 - 2
6	17	13	12	2	0 - 3
2	17	13	12	6	0 - 4
2	13	17	12	6	1 - 2
2	12	17	13	6	1 - 3
2	6	17	13	12	1 - 4
2	6	13	17	12	2 - 3
2	6	12	17	13	2 - 4
2	6	12	13	17	3 - 4

Analysis Of Selection Sort

```
selectionsort ( int  *a, int  n )
{
     for ( i = 0 ; i < n - 1 ; i + + )
     {
          for ( j = i + 1 ; j < n ; j + + )
          {
               if ( a[ i ] > a[ j ] )
               {
                    t = a[ i ] ;
                    a[ i ] = a[ j ] ;
                    a[ j ] = t ;
               }
          }
     }
}
```

17, 6, 13, 12, 2

i	No. of comp.
0	4
1	3
2	2
3	1

$$\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$$

$$= O(N^2)$$

Bubble Sort

```
main( )
{
   int  a[ ] = { 17, 6, 13, 12, 2 } ;
   int  i, j, t ;
   for ( j = 0 ; j <= 3 ; j + + )
   {
     for ( i = 0 ; i <= 3 - j ; i + + )
     {
       if ( a[ i ] > a[ i + 1 ] )
       {
          t = a[ i ] ;  a[ i ] = a[ i + 1 ] ;
          a[ i + 1 ] = t ;
       }
     }
   }
   for ( i = 0 ; i <= 4 ; i + + )
      printf ( "%d", a[ i ] ) ;
}
```

17	6	13	12	2		i i+1
6	17	13	12	2		0 - 1
6	13	17	12	2		1 - 2
6	13	12	17	2		2 - 3
6	13	12	2	17		3 - 4
6	13	12	2	17		0 - 1
6	12	13	2	17		1 - 2
6	12	2	13	17		2 - 3
6	12	2	13	17		0 - 1
6	2	12	13	17		1 - 2
2	6	12	13	17		0 - 1

Analysis Of Bubble Sort

```
bubblesort ( int  *a, int  n )
{
    for ( j = 0 ; j < n - 1 ; j + + )
    {
        for ( i = 0 ; i < ( n - 1 ) - j ; i + + )
        {
            if ( a[ i ] > a[ i + 1 ] )
            {
                t = a[ i ] ;
                a[ i ] = a[ i + 1 ] ;
                a[ i + 1 ] = t ;
            }
        }
    }
}
```

17, 6, 13, 12, 2

j	No. of comp.
0	4
1	3
2	2
3	1

$$\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$$

$$= O(N^2)$$

Radix Sort

9	47	21	32	5	13	27	4	54	76	29	85	98	62	30

Q_0	30				
Q_1	21				
Q_2	32	62			
Q_3	13				
Q_4	4	54			
Q_5	5	85			
Q_6	76				
Q_7	47	27			
Q_8	98				
Q_9	9	29			

Add elements in respective Q. initially, w.r.t units place then w.r.t tens place and so on....

30	21	32	62	13
4	54	5	85	76
47	27	98	9	29

Q_0	4	5	9		
Q_1	13				
Q_2	21	27	29		
Q_3	30	32			
Q_4	47				
Q_5	54				
Q_6	62				
Q_7	76				
Q_8	85				
Q_9	98				

4	5	9	13	21	27	29	30	32	47	54	62	76	85	98

Program

```
main( )
{
    int  a[ ] = { 9, 47, 21, 32, 5,
                  13, 27, 4, 54, 76 } ;
    int  arr[ 10 ][ 3 ], k, dig  ;
    dig = 1 ;
    for ( k = 0 ; k <= 1 ; k + + )
    {
       initq ( arr ) ;
       radix ( a, arr, 10, dig ) ;
       combine ( a, arr ) ;
       dig *= 10 ;
    }
    for ( i = 0 ; i < 10 ; i + + )
       printf ( "%d", a [ i ] ) ;
}
```

```
initq ( int  arr[ 10 ][ 3 ] )
{
    int  i, j ;
    for ( i = 0 ; i < 10 ; i + + )
    {
        for ( j = 0 ; j < 3 ; j + + )
           arr[ i ][ j ] = 0 ;
    }
}
```

Contd...

..Contd.

```
radix ( int  a[ ], int  arr[ ][ 3],
   int  n, int  dig )
{
   int  i, j, key ;
   for ( i = 0 ; i < n ; i + + )
   {
      key = ( a [ i ] / dig ) % 10 ;
      for ( j = 0 ; j < 3 ; j + + )
      {
         if ( arr[ key ][ j ] == 0 )
         {
            arr[ key ][ j ] = arr[ i ] ;
            break ;
         }
      }
   }
}
```

```
combine ( int  *a, int arr[ ][ 3 ] )
{
   int  i, j, x = 0 ;
   for ( i = 0 ; i < 10 ; i + + )
   {
      for ( j = 0 ; j < 3 ; j + + )
      {
         if ( arr[ i ][ j ] ! = 0 )
         {
            a[ x ] = arr[ i ][ j ] ;
            x + + ;
         }
      }
   }
}
```

Sorting & Recursion

Asang Dani

Objectives

- Insertion Sort
- Recursive Functions

Insertion Sort

```
main( )
{
   int  a[ ] = { 10, 12, 18, 19, 24, 2 } ;
   int   i, j, k, t ;
   for ( i = 1 ; i <= 5 ; i ++ )
   {
      t = a[ i ] ;
      for ( j = 0 ; j < i ; j++ )
      {
         if ( t < a[ j ] )
         {
            for ( k = i ; k >= j ; k-- )
               a[ k ] = a[k - 1] ;
               a[ j ] = t ;
               break ;
         }
      }
   }
```

```
   for ( i = 0 ; i <= 5 ; i ++ )
      printf ( "%d\t", a[ i ] ) ;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
10	12	18	19	24	2

t

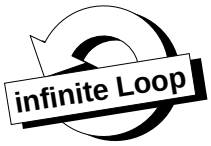
2

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
	10	12	18	19	24

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
2	10	12	18	19	24

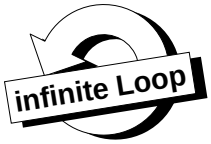
Simple Form

```
main( )
{
    printf ( ”Hi” ) ;
    main( ) ;
}
```



One More Form

```
main( )
{
    f( ) ;
}
f( )
{
    printf ( ”Hi” ) ;
    f( ) ;
}
```



More General

```
main( )
{
    int  num, sum ;
    printf ( ”Enter a number” ) ;
    scanf ( ”%d”, &num ) ;
    sum =  sumdig ( num ) ;
    printf ( ”%d”, sum ) ;
}
sumdig ( int  n )
{
    int  d ; int  s = 0 ;
    while ( n != 0  )
    {
        d = n % 10 ;
        n = n / 10 ;   s = s + d ;
    }
    return ( s ) ;
}
```

31698 → d5

327

485 → d3

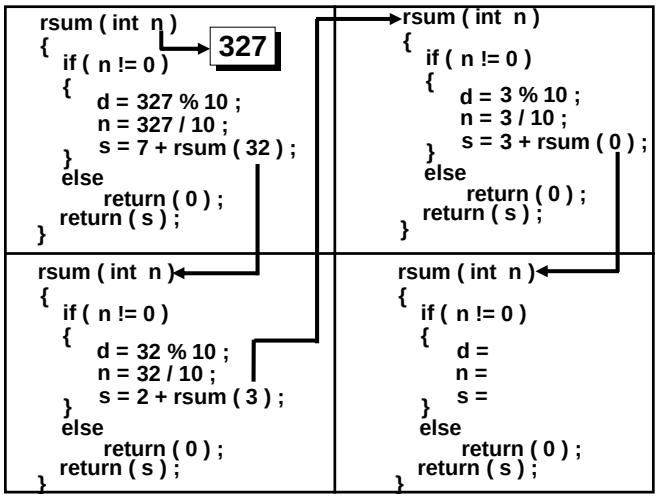
12

n	s	d
327	0	7
32	7	2
3	9	3
0	12	

```
main( )
{
    int  num, sum ;
    printf ( ”Enter a number” ) ;
    scanf ( ”%d”, &num ) ;
    sum =  rsum ( num ) ;
    printf ( ”%d”, sum ) ;
}
rsum ( int  n )
{
    int  d ; int  s ;
    if ( n != 0 )
    {
        d = n % 10 ;  n = n / 10 ;
        s = d + rsum ( n ) ;
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

327

4!
4 * 3 * 2 * 1
4 * 3 !
3 * 2 !
2 * 1 !



Recursive Factorial

```
main( )
{
    int  num, fact ;
    printf ( "Enter no." ) ;
    scanf ( "%d", &num ) ;
    fact = refact ( num ) ;
    printf ( "%d", fact ) ;
}
refact ( int  n )
{
    int  p ;
    if (  n != 0  )
       p = n * refact ( n - 1 ) ;
    else
       return ( 1 ) ;
    return ( p ) ;
}
```

Recursion Tips

- Make recursive call in an if
- else block is escape route
- else contains end cond. logic
- return may not be present

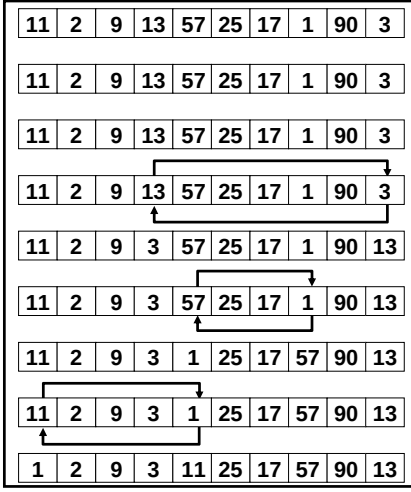
QuickSort

Asang Dani

Objectives

- Splitting mechanism in quick sort
- QuickSort algorithm
- QuickSort program

Split Array



Quick Sort

```
void quicksort ( int *, int, int ) ;
int split ( int *, int, int ) ;
main( )
{
    int arr[ 10 ] = { 11, 2, 9, 13, 57, 25, 17, 1, 90, 3 } ;
    int i ;
    quicksort ( arr, 0, 9 ) ;
    for ( i = 0 ; i <= 9 ; i++ )
        printf ( "%d\t", arr[ i ] ) ;
}
void quicksort ( int *a, int lower, int upper )
{
    int i ;
    if ( upper > lower )
    {
        i = split ( a, lower, upper ) ;
        quicksort ( a, lower, i - 1 ) ;
        quicksort ( a, i + 1, upper ) ;
    }
}
```

Cont…

Cont… | 11 | 2 | 9 | 13 | 57 | 25 | 17 | 1 | 90 | 3 |

p - L to R
q - R to L

```
int split ( int a[ ], int lower, int upper )
{
    int i, p, q, t ;
    p = lower + 1 ;
    q = upper ;
    i = a[ lower ] ;

    while ( p <= q )
    {
        while ( a[ p ] < i )
            p++ ;
        while ( a[ q ] > i )
            q-- ;
        if ( p < q )
        {
            t = a[ p ] ;
            a[ p ] = a[ q ] ;
            a[ q ] = t ;
        }
    }
    t = a[ lower ] ;
    a[ lower ] = a[ q ] ;
    a[ q ] = t ;
    return q ;
}
```

11	2	9	3	1	25	17	57	90	13

1	2	9	3	11	25	17	57	90	13

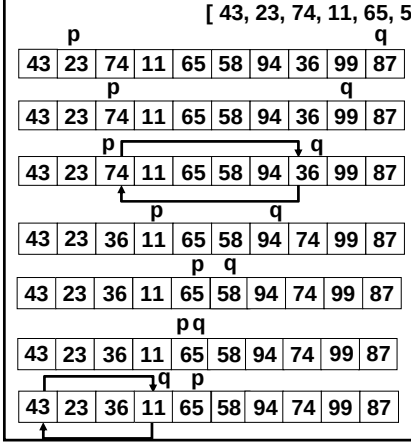
Problem

Show the steps for sorting the following numbers using Quick Sort Algorithm

[42, 23, 74, 11, 65, 58, 94, 36, 99, 87]

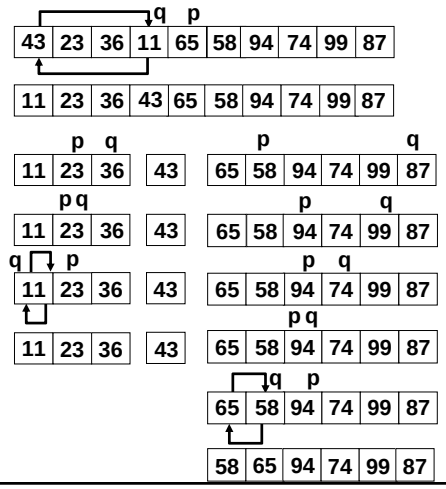
Exercise

Write the steps for quick sort of following elements:
[43, 23, 74, 11, 65, 58, 94, 36, 99, 87]



Contd...

Contd…



Complexity

Algorithm	Worst Case	Best Case
Bubble sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	$\log_2 n$
Insertion sort	$O(n^2)$	$n - 1$
Binary Tree Sort	$O(n^2)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$

Structures

Asang Dani

Objectives

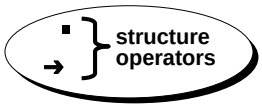
- Defining and using structures
- Creating an array of structures
- How to copy one structure variable into another
- Using nested structures
- Passing structure elements
- Passing structures

Handling Data

```
main( )
{
    char  n[ ] = { 'A', 'X', 'Y', '\0' } ;
    int  a[ ] = { 23, 27, 28 } ;
    float  s[ ] = { 4000.50, 5000.00, 6000.75 } ;
    int  i ;
    for ( i = 0  ; i <= 2 ; i ++ )
        printf ( "%c %d %f", n[ i ], a[ i ], s[ i ] ) ;
}
```

Structures

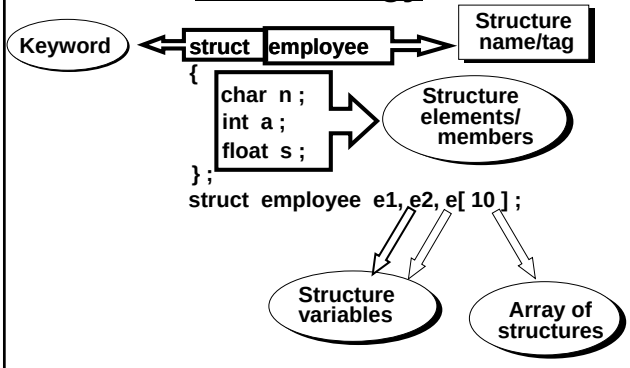
```
main( )
{
    struct  employee
    {
        char  n ;
        int  a ;
        float  s ;
    } ;
    struct  employee  e1 = { 'A', 23, 4000.50 } ;
    struct  employee  e2 = { 'X', 27, 5000.00 } ;
    struct  employee  e3 = { 'Y', 28, 6000.75 } ;
    printf ( "%c %d %f", e1.n, e1.a, e1.s ) ;
    printf ( "%c %d %f", e2.n, e2.a, e2.s ) ;
    printf ( "%c %d %f", e3.n, e3.a, e3.s ) ;
}
```



Array of Structures

```
main( )
{
    struct employee
    {
        char  n ;
        int  a ;
        float  s ;
    } ;
    struct  employee  e[ ] = {
                                { 'A', 23, 4000.50 } ,
                                { 'X', 27, 5000.00 } ,
                                { 'Y', 28, 6000.75 }
                              } ;
    int  i ;
    for (  i = 0 ;  i <= 2 ;  i++ )
        printf ( "%c %d %f", e[ i ] .n,  e[ i ].a, e[ i ].s ) ;
}
```

Terminology



Conclusion

- A structure is usually a collection of dissimilar elements.
- Structure elements are always stored in adjacent memory locations.

struct employee e[3] = { .. } ;

A	23	400.50	X	27	500.00	Y	28	600.75
401			408			415		

Array of Structures

A	23	400.50	X	27	500.00	Y	28	600.75
401			408			415		

```
struct employee e[ ] = { ... } ;
char *p ;
struct employee *q ;
struct employee (*r )[ 3 ] ;

p = e ; q = e ; r = e ;
p++ ; q++ ; r++ ;
printf ( "%u", p ) ;
printf ( "%u", q ) ;
printf ( "%u", r ) ;
```

Ptr. to structure

Ptr. to array of structures

402

408

422

```
struct employee *z[3] ;
```

Array of Ptrs to structures

Copying

```
main( )
{
    struct emp
    {
        char n[20] ;
        int a ;
        float s ;
    } ;
    struct emp e1 = { "Rahul", 23, 4000.50 } ;
    struct emp e2, e3 ;
    e2.n = e1.n ;
    e2.a = e1.a ;
    e2.s = e1.s ;
    e3 = e1 ;
    printf ( "%s %d %f", e3.n, e3.a, e3.s ) ;
}
```

e1

Rahul	23	400.50

e2

Rahul	23	400.50

e3

Rahul	23	400.50

piecemeal copying

strcpy (e2.n, e1.n) ;

copying at one shot

Nested Structures

```
main( )
{
   struct  address
   {
      char  city[20] ;
      long  int  pin ;
   } ;
   struct emp
   {
      char  n[20] ; int  age ;
       struct  address  a ; float  s ;
    } ;
   struct emp e={ "Rahul", 23, "Ngp", 44010,  4000.50 } ;
   printf ( "%s %d  %s %ld  %f", e.n,  e.age,  e.city,  e.pin,
              e.s ) ;
}
```

e

a

Rahul	23	Ngp	44010	400.50

e.a.city a.city e.a.pin

printf (”%d”, a.b.c.d.e.f) ; f - int

Passing Structure Elements

```
main( )
{
    struct  book
    {
        char  n[20] ; int  nop ; float  pr ;
    } ;
    struct  book  b = { "Basic", 425, 135.00 } ;
    display (b.n,  b.nop,  b.pr ) ;
    show ( b.n,  &b.nop, &b.pr ) ;
}
display ( char *n, int  pg, float  p )
{
    printf ( "%s %d %f", n, pg, p ) ;
}
show ( char *n, int *pg, float *p )
{
    printf ( "%s %d %f",  n,  *pg, *p ) ;
}
```

b

Basic	425	135.00
501	521	523

Passing Structures

```
main( )
{
   struct  book
   {
     char  n[20] ; int  nop ;  float  pr ;
   } ;
   struct book b = { "Basic", 425, 135.00 } ;
   display1 ( b ) ; show1 ( &b ) ;
}
display1 (struct  book  bb )
{
   printf ("%s %d %f",  bb.n, bb.nop, bb.pr ) ;
}
show1 ( struct  book  *bb )
{
   printf ( "%s %d %f", ( *bb ).n,( *bb ).nop,( *bb ).pr );
   printf ( "%s %d %f",   bb -> n, bb -> nop, bb -> pr ) ;
}
```

b

Basic	425	135.00
501	521	523

Structures & Polynomials

Asang Dani

Objectives

- Representing polynomials using structures
- Passing array of structures
- How to perform operations like addition and multiplication on polynomials using structures

Problem

- There are two arrays A & B. A contains 25 elements whereas B contains 30 elements. Write a procedure to create an array C which contains only those elements which are common to A & B.

Polynomial

$X^7 + 2X^6 + 3X^5 + 4X^4 + 5X^2$

```
int a[ ] = { 1, 7, 2, 6, 3, 5, 4, 4, 5, 2 } ;
```

Problems

- Only integer coefficients
- Not intuitive - What is a[7]

```
struct term
{
  int coeff ;
  int exp ;
} ;
```

a[ ]

1	7	2	6	3	5	4	4	5	2

```
struct term a[ ] = { 1, 7, 2, 6, 3, 5, 4, 4, 5, 2 } ;
printf ( "%d %d", a[3].coeff, a[3].exp ) ;
```

Passing Array of Structures

```
struct term
{
    int coeff ; int exp ;
} ;
main( )
{
    struct term a[ ] = { 1, 7, 2, 6, 3, 5, 4, 4, 5, 2 } ;
    fun ( a ) ;
}
fun ( struct term *p )
{
    int i ;
    for ( i = 0 ; i < 5 ; i++ )
    {
        printf ( "%d %d", ( * ( p + i ) ).coeff, ( *( p + i ) ).exp ) ;
        printf ( "%d %d", p[ i ].coeff, p[ i ].exp ) ;
    }
}
```

a[ ]

1	7	2	6	3	5	4	4	5	2

p p+1 p+2

Polynomial Add^n

$X^7 + 2X^6 + 3X^5 + 4X^4 + 5X^2$

$X^4 + X^3 + X^2 + X + 2$

```
struct term
{
  int coeff ;
  int exp ;
} ;
```

a[ ]

1	7	2	6	3	5	4	4	5	2

b[ ]

1	4	1	3	1	2	1	1	2	0

```
int polyadd ( struct term *, int, struct term *, int, struct term * ) ;

main( )
{
  struct term a[ ] = { 1, 7, 2, 6, 3, 5, 4, 4, 5, 2 } ;
  struct term b[ ] = { 1, 4, 1, 3, 1, 2, 1, 1, 2, 0 } ;
  struct term c[20] ;
  int numa = 5, numb =  5, numc, i ;

  numc = polyadd ( a, numa, b, numb, c ) ;
```

Cont…

Cont…

```
  for ( i = 0 ; i < numc ; i++ )
      printf ( "%d x^%d + ", c[ i ].coeff, c[ i ].exp ) ;
}

int polyadd ( struct term *pa, int na, struct term *pb, int nb,
                                                  struct term *pc )
{
    int i = 0, j = 0, k = 0 ;
    while ( i < na && j < nb )
    {
        if ( pa[ i ].exp == pb[ j ] .exp )
        {
            pc[ k ].coeff = pa[ i ].coeff + pb[ j ] .coeff ;
            pc[ k ].exp = pa[ i ].exp ;
            i++ ;  j++ ;  k++ ;
        }
```

Cont…

Cont… $X^7 + 2X^6 + 3X^5 + 4X^4 + 5X^2$ $X^4 + X^3 + X^2 + X + 2$

```
  else
  {
      if ( pa[ i ].exp > pb[ j ].exp )
      {
        pc[ k ].coeff = pa[ i ].coeff ;
        pc[ k ].exp = pa[ i ].exp ;
        i++ ;  k++ ;
      }
      else
      {
        pc[ k ]  = pb[ j ] ;
        j++ ;  k++ ;
      }
  }
} // end of while loop

  while ( i < na )
  {
    pc[ k ] = pa[ i ] ;
    i++ ;  k++ ;
  }

  while ( j < nb )
  {
    pc[ k ] = pb[ j ] ;
    j++ ;  k++ ;
  }

  return k ;
}
```

pc[k] = pa[i] ;

Polynomial Multiplication

```
struct  term
{
  int coeff ;
  int exp ;
} ;
int polymul ( struct term *, int, struct term *, int, struct term * ) ;
int polyadd ( struct term *, int, struct term *, int, struct term * ) ;
main( )
{
  struct term a[ ] = { 1, 7, 2, 6, 3, 5, 4, 4, 5, 2 } ;
  struct term b[ ] = { 1, 4, 1, 3, 1, 2, 1, 1, 2, 0 } ;
  struct term c[20] ;
  int numa = 5, numb =  5, numc, i ;
  numc = polymul ( a, numa, b, numb, c ) ;
  for ( i = 0 ; i < numc ; i++ )
      printf ( "%d x^%d + ", c[ i ].coeff, c[ i ].exp ) ;
}
```

Cont…

```
int polymul ( struct term *pa, int na, struct term *pb, int nb,
                                                struct term *pc )
{
    struct term t , temp[20] ;
    int  i, j, numpc, numtemp = 0, k ;
    for ( i = 0 ; i < na ; i++ )
    {
        for ( j = 0 ; j < nb ; j++ )
        {
            t.coeff = pa[ i ].coeff * pb[ j ].coeff ;
            t.exp = pa[ i ].exp + pb[ j ].exp ;
            numpc = polyadd ( &t, 1, temp, numtemp, pc ) ;
            for ( k = 0 ; k < numpc ;  k++ )
                temp[ k ] = pc[ k ] ;
            numtemp = numpc ;
        }
    }
    return  numpc ;
}
```

$X^7 + 2\,X^6 + 3\,X^5 + 4\,X^4 + 5\,X^2$

$X^4 + X^3 + X^2 + X + 2$

Solution

Iteration	t	temp	pc
1	x^{11}	Empty	x^{11}
2	x^{10}	Empty	x^{10}

Two Dimensional Arrays

Asang Dani

Objectives

- Declaring and using two dimensional arrays
- How to write a program to find the saddle point

Two Dimensional Array

```
main( )
{
    int a[ ][ 5 ] =    {
                           { 2, 6, 1, 8, 4 },
                           { 1, 2, 5, 6, 8 },
                           { 7, 9, 8, 7, 21 },
                           { 4, 5, 6, 8, 10 }
                       } ;
    int  i, j ;
    printf ( "%d", a[ 2 ][ 4 ] ) ;
    printf ( "%d %d", sizeof ( a ), a ) ;
    for ( i = 0 ; i <= 3 ; i + + )
    {
          for ( j = 0 ; j <= 4 ; j + + )
              printf ( "%d", a [ i ][ j ] ) ;
          printf ( "\n" ) ;
    }
}
```

optional

compulsory

21

optional

int b[ ][1][2][3]

compulsory

40 4080

Saddle Point

```
main( )
{
  int a[ ][ 5 ] = {
                    5, 2, 2, 6, 5,
                    4, 9, 3, 4, 8,
                    9, 4, 2, 1, 9,
                    7, 1, 0, 8, 7,
                    6, 2, 1, 5, 9
                  } ;
  for ( i = 0 ; i <= 4 ; i++ )
  {
     small = a[ i ][ 0 ] ;
     for ( j = 0 ; j <= 4 ; j++)
     {
          if ( a[ i ][ j ] < small )
          {
               small = a[ i ][ j ] ;
               col = j ;
          }
     }
     big = small ;
     for ( r = 0 ; r <= 4 ; r++ )
     {
        if ( a[ r ][ col ] > big )
        {
           big = a[ r ][ col ] ;
           row = r ;
        }
     }
     if ( big == small )
     {
        printf ( "%d", big ) ;
        printf ( "%d %d", row, col );
        break ;
     }
  }
}
```

Define variables

Problem

- Given 2 matrices A & B of the order (n x n)
- Upper triangle = 0, Lower triangle = Non-zero
- Generate C of n x (n + 1) t o accommodate A and B

a[4][4]

1	0	0	0
5	2	0	0
6	7	3	0
8	9	10	4

1	11	15	16	18
5	2	12	17	19
6	7	3	13	20
8	9	10	4	14

c[4][5]

11	0	0	0	
15	12	0	0	
16	17	13	0	
18	19	20	14	

b[4][4]

A	C
0, 0	0, 0
1, 0	1, 0
2, 0	2, 0
3, 0	3, 0
1, 1	1, 1
2, 1	2, 1
3, 1	3, 1
2, 2	2, 2
3, 2	3, 2
3, 3	3, 3

B	C
0, 0	0, 1
1, 0	0, 2
2, 0	0, 3
3, 0	0, 4
1, 1	1, 2
2, 1	1, 3
3, 1	1, 4
2, 2	2, 3
3, 2	2, 4
3, 3	3, 4

```
main( )
{
   int a[ ][ 4 ] = {
                     1, 0, 0, 0,
                     5, 2, 0, 0,
                     6, 7, 3, 0,
                     8, 9, 10, 4
                   } ;
   int b[ ][ 4 ] = {
                     11, 0, 0, 0,
                     15, 12, 0, 0,
                     16, 17, 13, 0,
                     18, 19, 20, 14
                   } ;
   int c[ 4 ][ 5 ], i, j ;
   for ( j = 0 ; j <= 3 ; j++ )
   {
      for ( i = j ;  i <= 3 ; i++ )
         c[ i ][ j ] = a[ i ][ j ] ;
   }
```

A	C
0, 0	0, 0
1, 0	1, 0
2, 0	2, 0
3, 0	3, 0
1, 1	1, 1
2, 1	2, 1
3, 1	3, 1
2, 2	2, 2
3, 2	3, 2
3, 3	3, 3

B	C
0, 0	0, 1
1, 0	0, 2
2, 0	0, 3
3, 0	0, 4
1, 1	1, 2
2, 1	1, 3
3, 1	1, 4
2, 2	2, 3
3, 2	2, 4
3, 3	3, 4

```
   for ( i = 0 ; i <= 3 ; i ++ )
   {
     for ( j = i + 1 ; j <= 4 ; j++ )
        c[ i ][ j ] = b[ j - 1 ][ i ] ;
   }
```

Cont…

Cont…

```
    for ( i = 0 ; i <= 3 ; i ++ )
    {
      for ( j = 0 ;  j <= 4 ; j++ )
            printf ( "%d ", c[ i ][ j ] ) ;
      printf ( "\n" ) ;
    }
}
```

Determine

Determine values of a[i][j] and b[i][j] from matrix c

Determine a[i][j]

```
if ( j > i )
   element = 0 ;
else
   element = c[ i ] [ j ] ;
```

a[3][2] = c[3][2]

Determine b[i][j]

```
if ( j > i )
   element = 0 ;
else
   element = c[ j ] [ i + 1 ] ;
```

b[3][2] = c[2][4]

1	0	0	0
5	2	0	0
6	7	3	0
8	9	10	4

4 x 4

11	0	0	0
15	12	0	0
16	17	13	0
18	19	20	14

4 x 4

→

1	11	15	16	18
5	2	12	17	19
6	7	3	13	20
8	9	10	4	14

4 x 5

Problem

➡ A magic square of 4 rows x 4 columns contains different elements. Write a function to verify whether the sum of each individual column elements, sum of each individual row elements and sum of diagonal elements is equal or not.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

34

Sparse Matrices

Asang Dani

Objectives

- Sparse matrices
- How to pass arrays
- Performing 3 tuple conversion

Sparse Matrices

15	0	0	22	0	-15
0	11	3	0	0	0
0	0	0	- 6	0	0
0	0	0	0	0	0
91	0	0	0	0	0
0	0	28	0	0	0

3-Tuple Form

r	c	Non-zero
6	6	8
0	0	15
0	3	22
0	5	-15
1	1	11
1	2	3
2	3	- 6
4	0	91
5	2	28

Passing Arrays

```
main( )
{
    int a[ ][ 4 ] = {
                        1, 0, 3, 8,
                        5, 2, 0, 7
                    } ;
    fun1 ( a, 2, 5 ) ;
    fun2 ( a, 2, 5 ) ;
}
fun1 ( int ( *p )[ 4 ], int r, int c )
{
   int i, j ;
   for ( i = 0 ; i < r ; i++ )
   {
      for ( j = 0 ;  j < c ; j++ )
         printf ( "%d", p[ i ][ j ] ) ;
   }
}
```

```
fun2 ( int *pa, int  r, int c )
{
   int i, j ;
   for ( i = 0 ; i < r ; i++ )
   {
      for ( j = 0 ;  j < c ; j++ )
      {
         printf ( "%d", *pa ) ;
         pa++ ;
      }
   }
}
```

General

a[ ]

1	0	3	8	5	2	0	7

p[1][3] → * (* (p + 1) + 3)

3-Tuple Conversion

```
# include <alloc.h>
main( )
{
    int  a[ 3 ][ 4 ] = {
                          4, 0, 0, 1,
                          2, 0, 0, 9,
                          6, 1, 0, 0
                        } ;
    int  *ta ;
    ta = create ( a, 3, 4 ) ;
    display ( ta ) ;
    free ( ta ) ;
}
```

r	c	Non-zero
3	4	6
0	0	4
0	3	1
1	0	2
1	3	9
2	0	6
2	1	1

Cont…

```
int * create ( int  *pa, int  r, int c )
{
    int  rows, *p, i, j, k ;
    rows = count ( pa, r, c ) + 1 ;
    p = ( int * ) malloc ( rows *
                  3 * sizeof ( int ) ) ;
    p[ 0 ] = r ; p[ 1 ] = c ;
    p[ 2 ] = rows - 1; k = 3 ;
```

```
    for ( i = 0 ; i < r ; i++ )
    {
       for ( j = 0 ; j < c ; j++ )
       {
          if ( *pa != 0 )
          {
             p[ k ] = i ;  k++ ;
             p[ k ] = j ;  k++ ;
             p[ k ] = *pa ;  k++ ;
          }
          pa ++ ;
       }
    }
    return p ;
}
```

pa

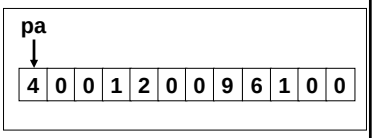
4	0	0	1	2	0	0	9	6	1	0	0

p

3	4	6	0	0	4	0	3	1	1	0	2	1	3	9	2	0	6	2	1	1

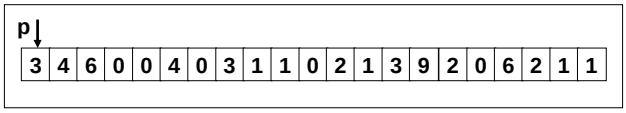
Cont…

```
int count ( int  *pa, int  r, int  c )
{
   int  count = 0, i, j ;
   for ( i = 0 ; i < r ; i++ )
   {
      for ( j = 0 ; j < c ; j++ )
      {
         if ( *pa != 0 )
            count ++ ;

         pa++ ;
      }
   }
   return  count ;
}
```



Cont…

```
void display ( int *p )
{
  int  i, rows ;
  rows = p[ 2 ] + 1 ;
  for ( i = 0 ; i < rows  * 3 ; i++ )
  {
    if ( i % 3 == 0 )
         printf ( "\n" ) ;
    printf ( "%d\t", p[ i ] ) ;
  }
}
```



Transpose of Sparse Matrices

Asang Dani

Objectives

- What is a transpose?
- How to get the transpose of a sparse matrix

Transpose

$$\begin{bmatrix} 4 & 0 & 0 & 3 \\ 3 & 0 & 0 & 1 \\ 0 & 0 & 2 & 5 \\ 11 & 0 & 0 & 0 \\ 12 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 & 0 & 11 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 3 & 1 & 5 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 4 & 9 \\ 0 & 0 & 4 \\ 0 & 3 & 3 \\ 1 & 0 & 3 \\ 1 & 3 & 1 \\ 2 & 2 & 2 \\ 2 & 3 & 5 \\ 3 & 0 & 11 \\ 4 & 0 & 12 \\ 4 & 3 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 5 & 9 \\ 0 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 3 & 11 \\ 0 & 4 & 12 \\ 2 & 2 & 2 \\ 3 & 0 & 3 \\ 3 & 1 & 1 \\ 3 & 2 & 5 \\ 3 & 4 & 1 \end{bmatrix}$$

Hint
Search for 0, 1, 2, etc. in first column of matrix A and then set matrix B

Program - Transpose

```
#include <alloc.h>
main( )
{
  int a[ 5 ][ 4 ] = {
                      4, 0, 0, 3,
                      3, 0, 0, 1,
                      0, 0, 2, 5,
                      11, 0, 0, 0,
                      12, 0, 0, 1
                    } ;
  int  *ta, *tb ;
  ta = create ( a, 5, 4 ) ;
  tb = transpose ( ta ) ;
  display ( tb ) ;
  free ( ta ) ;
  free ( tb ) ;
}
```

5	4	9	4	5	9
0	0	4	0	0	4
0	3	3	0	1	3
1	0	3	0	3	11
1	3	1	0	4	12
2	2	2	2	2	2
2	3	5	3	0	3
3	0	11	3	1	1
4	0	12	3	2	5
4	3	1	3	4	1

```
int * transpose ( int *ta )
{
  int  rows, c, nz, p, q, cols ;
  int  *tb ;
  rows = ta[ 2 ] + 1 ;
  tb = ( int * ) malloc ( rows *
            3 * sizeof ( int ) ) ;
  tb[ 0 ] = cols = ta[ 1 ] ;
  tb[ 1 ] = ta[ 0 ] ;
  tb[ 2 ] = nz = ta[ 2 ] ;
  q = 1 ;
  for (c = 0 ; c < cols ; c++ )
  {
    for ( p = 1; p <= nz; p++)
    {
      if ( ta[ p * 3 + 1 ] == c )
      {
        tb[ q*3 ] = ta[ p*3+1 ] ;
        tb[ q*3+1 ] = ta[ p*3 ] ;
        tb[ q*3+2 ] = ta[ p*3+2 ] ;
        q++ ;
      }
    }
  }
  return  tb ;
}
```

5	4	9	4	5	9
0	0	4	0	0	4
0	3	3	0	1	3
1	0	3	0	3	11
1	3	1	0	4	12
2	2	2	2	2	2
2	3	5	3	0	3
3	0	11	3	1	1
4	0	12	3	2	5
4	3	1	3	4	1

Problem

Write a program to verify whether transpose of a given sparse matrix is same as the original sparse matrix.

Addition of Sparse Matrices

Asang Dani

Objectives

- Sparse Matrices
- Perform addition of two sparse matrices

Addition of SM

$$\begin{bmatrix} 4 & 0 & 0 & 3 \\ 3 & 0 & 0 & 1 \\ 0 & 0 & 2 & 5 \\ 11 & 0 & 0 & 0 \\ 12 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 5 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 11 & 0 & 0 & 3 \\ 3 & 2 & 1 & 1 \\ 0 & 5 & 2 & 2 \\ 11 & 0 & 0 & 0 \\ 13 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 4 & 9 \\ 0 & 0 & 4 \\ 0 & 3 & 3 \\ 1 & 0 & 3 \\ 1 & 3 & 1 \\ 2 & 2 & 2 \\ 2 & 3 & 5 \\ 3 & 0 & 11 \\ 4 & 0 & 12 \\ 4 & 3 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 4 & 6 \\ 0 & 0 & 7 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 5 \\ 2 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 & 12 \\ 0 & 0 & 11 \\ 0 & 3 & 3 \\ 1 & 0 & 3 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 3 & 1 \\ 2 & 1 & 5 \\ 2 & 2 & 2 \\ 2 & 3 & 2 \\ 3 & 0 & 11 \\ 4 & 0 & 13 \\ 4 & 3 & 1 \end{bmatrix}$$

Tips

5	4	9		5	4	6		5	4	12
0	0	4		0	0	7		0	0	11
0	3	3		1	1	2		0	3	3
1	0	3	+	1	2	1	=	1	0	3
1	3	1		2	1	5		1	1	2
2	2	2		2	3	2		1	2	1
2	3	5		4	0	1		1	3	1
3	0	11						2	1	5
4	0	12						2	2	2
4	3	1						2	3	2
								3	0	11
								4	0	13
								4	3	1

- Rows in C = Rows in A + Rows in B - Sometimes True
- Row of A < Row of B - Copy from A
- Row of B < Row of A - Copy from B
- Row A = Row B Check columns

Addition of SM

```
# include  <alloc.h>
main( )
{
  int a[ 5 ][ 4 ] = {
                       4, 0, 0, 3,
                       3, 0, 0, 1,
                       0, 0, 2, 5,
                       11, 0, 0, 0,
                       12, 0, 0, 1
                     } ;
  int b[ 5 ][ 4 ] = {
                       7, 0, 0, 0,
                       0, 2, 1, 0,
                       0, 5, 0, 2,
                       0, 0, 0, 0
                       1, 0, 0, 0
                     } ;
    int  *ta, *tb, *tc ;
    ta = create ( a, 5, 4 ) ;
    tb = create ( b, 5, 4 ) ;
    tc = add (  ta, tb ) ;
    display ( tc ) ;
    free ( ta ) ;
    free ( tb ) ;
    free ( tc ) ;
}
```

5	4	9		5	4	6
0	0	4		0	0	7
0	3	3		1	1	2
1	0	3	+	1	2	1
1	3	1		2	1	5
2	2	2		2	3	2
2	3	5		4	0	1
..	..	..				

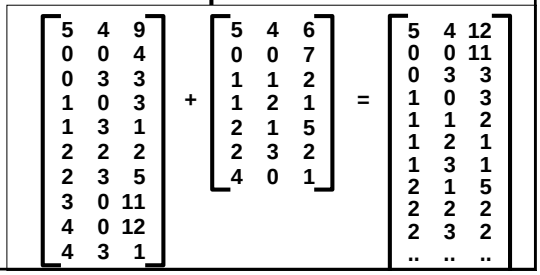
```
int * add ( int  *s1, int  *s2 )
{
  int *p,  i, j, k ;
  int  max, maxa, maxb ;
  int  rowa, cola, vala ;
  int  rowb, colb, valb ;

  maxa = s1[ 2 ] ; maxb = s2[ 2 ] ;
  max = maxa + maxb + 1 ;

  p = ( int * ) malloc ( max * 3 *
                         sizeof (int) ) ;
  i = j = k = 1 ;
  while ( k <= max )
  {
    if ( i <= maxa )
    {
      rowa =  s1[ i * 3 + 0 ] ;
      cola =  s1[ i * 3 + 1 ] ;
      vala =  s1[ i * 3 + 2 ] ;
    }
    else
      rowa = cola = BIGNUM ;
  #define BIGNUM 100
  if ( j <= maxb )
  {
    rowb = s2[ j * 3 + 0 ] ;
    colb = s2[ j * 3 + 1 ] ;
    valb = s2[ j * 3 + 2 ] ;
  }
  else
     rowb = colb = BIGNUM ;
```

5	4	9		5	4	6
0	0	4		0	0	7
0	3	3		1	1	2
1	0	3	+	1	2	1
1	3	1		2	1	5
2	2	2		2	3	2
2	3	5		4	0	1
..	..	..				

Cont…

```
if ( rowa < rowb )
{
  p[ k * 3 + 0 ] = rowa ;
  p[ k * 3 + 1 ] = cola ;
  p[ k * 3 + 2 ] = vala ;
  i++ ;
}
```

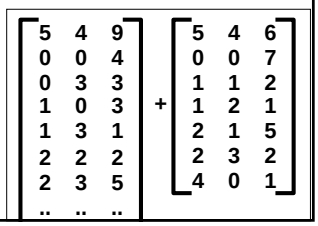
```
if ( rowa > rowb )
{
    p[ k * 3 + 0 ] = rowb ;
    p[ k * 3 + 1 ] = colb ;
    p[ k * 3 + 2 ] = valb ;
    j++ ;
}
```



Cont…

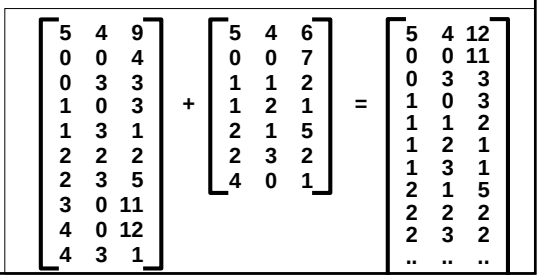
```
if ( rowa == rowb )
{
    if ( cola == colb )
    {
      p[ k*3+0 ] = rowa ;
      p[ k* 3+1 ] = cola ;
      p[ k*3+2 ] = vala + valb ;
      i++ ; j++ ; max- - ;
    }

    if ( cola < colb )
    {
      p[ k * 3 + 0 ] = rowa ;
      p[ k * 3 + 1 ] = cola ;
      p[ k * 3 + 2 ] = vala ;
      i++ ;
    }
```

```
    if ( cola > colb )
    {
      p[ k * 3 + 0 ] = rowb ;
      p[ k * 3 + 1 ] = colb ;
      p[ k * 3 + 2 ] = valb ;
      j++ ;
    }
  } // if
  k++ ;
} // end of while loop
```



Cont…

```
    p[ 0 ] = s1[ 0 ] ;
    p[ 1 ] = s1[ 1 ] ;
    p[ 2 ] = max ;

    return p ;
} // end of add( )
```



Multiplication of Sparse Matrices

Asang Dani

Objectives

- Matrix multiplication
- Multiplication of sparse matrices

Matrix Multiplication

$$\begin{bmatrix} 4 & 0 & 0 & 1 \\ 2 & 0 & 0 & 9 \\ 6 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 0 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 2 & 27 \\ 8 & 0 \end{bmatrix}$$

3 x 4 | 4 x 2 | 3 x 2

```
for ( i = 0 ; i < 3 ; i + + )
{
    for ( j = 0 ; j < 2 ; j + + )
    {
        s = 0 ;
        for ( k = 0 ;  k < 4 ; k + + )
            s = s + a[ i ][ k ] * b[ k ][ j ] ;
        c[ i ][ j ] = s ;
    }
}
```

i k	k j	i k	k j
00	00	00	01
01	10	01	11
02	20	02	21
03	30	03	31
10	00	10	01
11	10	11	11
12	20	12	21
13	30	13	31
20	00	20	01
21	10	21	11
22	20	22	21
23	30	23	31

Multiplication of SM

```
# include <alloc.h>
main( )
{
   int a[ 3 ][ 4 ] = {
                        4, 0, 0, 1,
                        2, 0, 0, 9,
                        6, 1, 0, 0
                      } ;

   int b[ 4 ][ 2 ] = {
                        1, 0,
                        2, 0,
                        0, 0,
                        0, 3
                      } ;

      int  *ta, *tb, *tc ;
      ta = create ( a, 3, 4 ) ;
      tb = create ( b, 4, 2 ) ;
      tc = mul ( ta, tb ) ;
      display ( tc ) ;
      free ( ta ) ;
      free ( tb ) ;
      free ( tc ) ;
  }
```

Cont…

```
int* mul ( int  *x, int  *y )
{
  int  *c, i , j, k, px ;
  k =  x[ 0 ] * y[ 1 ] + 1 ;
  z = ( int * ) malloc (   k * 3 *
                  sizeof ( int ) ) ;
  k = 1 ;
  for ( i = 0 ; i < x[ 0 ] ; i++ )
  {
    for ( j = 0 ; j < y[ 1 ] ; j++  )
    {
        px = s_in_x ( x, i ) ;
```

To be Continued...

```
s_in_x ( int  *p, int  i )
{
   int j ;
   for ( j = 1 ; j <= p[ 2 ] ; j++ )
   {
       if ( p[ j * 3 ] == i )
           return  j ;
       }
   }
   return  -1 ;
}
```

3	4	6
0	0	4
0	3	1
1	0	2
1	3	9
2	0	6
2	1	1

p

3	4	6	0	0	4	0	3	1	1	0	2	1	3	9	2	0	6	2	1	1

```
int* mul ( int  *x, int  *y )
{
  /* Existing Code */
  for ( i = 0 ; i < x[ 0 ] ; i++ )
  {
    for ( j = 0 ; j < y[ 1 ] ; j++  )
    {
        px = s_in_x ( x, i ) ;
        if ( px != -1 )
        {
           py = s_in_y ( y, j,
                    x[ px * 3+1 ] ) ;
```

4	0	0	1
2	0	0	9
6	1	0	0

1	0
2	0
0	0
0	3

```
s_in_y ( int  *p, int  j, int  colx )
{
  int i ;
  for ( i = 1 ; i <= p[ 2 ] ; i++ )
  {
    if ( p[ i * 3 + 1 ] == j &&
         p[ i * 3 ] == colx )
      return i ;
  }
  return  -1 ;
}
```

col X = row Y

3	4	6
0	0	4
0	3	1
1	0	2
1	3	9
2	0	6
2	1	1

4	2	3
0	0	1
1	0	2
3	1	3

```
int* mul ( int  *x, int  *y )
{
  for ( i = 0 ; i < x[ 0 ] ; i++ )
  {
    for ( j = 0 ; j < y[ 1 ] ; j++  )
    {
      px = s_in_x ( x, i ) ;
      if ( px != -1 )
      {
        s = 0 ;
        while ( x[ px * 3 ] == i )
        {
          py = s_in_y ( y, j, x[ px*3+1 ] );
          if ( py != -1 )
            s = s + x[ px * 3 + 2 ] * y[ py * 3 + 2 ] ;
          px++ ;
        }
        ..
}
```

4	0	0	1
2	0	0	9
6	1	0	0

1	0
2	0
0	0
0	3

3	4	6
0	0	4
0	3	1
1	0	2
1	3	9
2	0	6
2	1	1

4	2	3
0	0	1
1	0	2
3	1	3

```
int* mul ( int  *x, int  *y )
{
  int  *z, i, j, k, px, py, s ;
  for ( i = 0 ; i < x[ 0 ] ; i + + )
  {
    for ( j = 0 ; j < y[ 1 ] ; j + +  )
    {
      px = s_in_x ( x, i ) ;
      if ( px != -1 )
      {
        s = 0 ;
        while ( x[ px * 3 ] == i )
        {
          py = s_in_y ( ... ) ;
          if ( py != -1 )
            s = s + ... ;
          px++ ;
        }
        if ( s != 0 )
        {
          z[ k * 3 + 0 ] = i ;
          z[ k * 3 + 1 ] = j ;
          z[ k * 3 + 2 ] = s ;
          k++ ;
        }
      } // if
    } // j loop
  } //  i loop
  z[ 0 ] = x[ 0 ] ;
  z[ 1 ] = y[ 1 ] ;
  z[ 2 ] = k - 1 ;
  return  z ;
}
```

Storage

Asang Dani

Objectives

- How two dimensional arrays are stored
- 3-D arrays
- 4-D arrays
- N-D arrays

Storage - 2D Array

3 x 4 matrix

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}

Row major

a_{00}	a_{01}	a_{02}	a_{03}	a_{10}	a_{11}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{23}

int a[3][4] ;

a_{23} → BA + 2 * 4 + 3

int a[m][n]

a_{ij} → BA + i * n + j

Col major

a_{00}	a_{10}	a_{20}	a_{01}	a_{11}	a_{21}	a_{02}	a_{12}	a_{22}	a_{03}	a_{13}	a_{23}

int a[3][4] ;

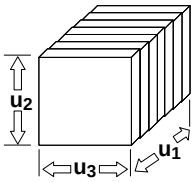
a_{23} → BA + 3 * 3 + 2

int a[m][n]

a_{ij} → BA + j * m + i

3-D Array

$a[\,1 \ldots u_1,\ 1 \ldots u_2,\ 1 \ldots u_3\,]$ ⇨



Row major

$a[\,i\,][\,j\,][\,k\,] \Rightarrow \alpha + (i - 1) * u_2u_3 + (j - 1) * u_3 + (k - 1)$

Column major

$a[\,i\,][\,j\,][\,k\,] \Rightarrow \alpha + (i - 1) * u_2u_3 + (k - 1) * u_2 + (j - 1)$

4-Dimensional Array

$a[\,1 \ldots u_1,\ 1 \ldots u_2,\ 1 \ldots u_3,\ 1 \ldots u_4\,]$

Row major

$a[\,i\,][\,j\,][\,k\,][\,l\,] \Rightarrow \alpha + (i - 1) * u_2u_3u_4 + (j - 1) * u_3u_4 + (k - 1) * u_4 + (l - 1)$

Column major

$a[\,i\,][\,j\,][\,k\,][\,l\,] \Rightarrow \alpha + (i - 1) * u_2u_3u_4 + (j - 1) * u_3u_4 + (l - 1) * u_3 + (k - 1)$

n-Dimensional Array

$a[\,1 \ldots u_1,\ 1 \ldots u_2,\ 1 \ldots u_3,\ \ldots,\ u_n\,]$

Row major

$a[\,i_1\,][\,i_2\,][\,i_3\,][\,..\,][\,i_n\,]$

$$\alpha + (i_1 - 1) * u_2u_3u_4 \ldots u_n + (i_2 - 1) * u_3u_4 \ldots u_n + (i_3 - 1) * u_4u_5 \ldots u_n + (i_4 - 1) * u_5u_6 \ldots u_n + \cdot + \cdot + \cdot + (i_{n-1} - 1) * u_n + (i_n - 1)$$

Column major

$a[\,i_1\,][\,i_2\,][\,i_3\,][\,..\,][\,i_n\,]$

$$\alpha + (i_1 - 1) * u_2u_3u_4 \ldots u_n + (i_2 - 1) * u_3u_4 \ldots u_n + (i_3 - 1) * u_4u_5 \ldots u_n + (i_4 - 1) * u_5u_6 \ldots u_n + \cdot + \cdot + \cdot + (i_n - 1) * u_{n-1} + (i_{n-1} - 1)$$

Problem

Find location of element A[6][2][3][8] relative to first element of the array A[3:8][2:4][3:6][6:9]

Row major

```
α +(6–3)*((4–2+1)*(6–3+1)*(9–6+1))
  +(2–2)*((6–3+1)*(9–6+1))
  +(3–3)*(9–6+1)
  + 8– 6
```

Col major

```
α +(6–3)*((4–2+1)*(6–3+1)*(9–6+1))
  +(2–2)*((6–3+1)*(9–6+1))
  +(8–6)*(6–3+1)
  + 3– 3
```

Dynamic Memory Allocation

Asang Dani

Objectives

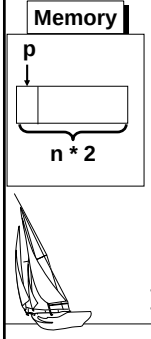
- Limitations on arrays
- Dynamic memory allocation
- How the memory allocation is done

Arrays

No Flexibility

```
main( )
{
    int  m1, m2, m3, i, per [ 10 ] ;
    for ( i = 0 ; i <= 9 ; i++ )
    {
        printf ( "Enter marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per [ i ] = ( m1 + m2 + m3 ) / 3 ;
        printf ( "%d", per [ i ] ) ;
    }
}
```

Dynamic Allocation




```
# include "alloc.h"
main( )
{
    int  *p ;  int  m1, m2, m3, i, per ;
    printf ( "Enter no. of students" ) ;
    scanf ( "%d", &n ) ;
    p = ( int * ) malloc ( n * 2  ) ;
    for ( i = 0 ; i < n ; i++ )
    {
        scanf ( "%d%d%d", &m1, &m2, &m3 ) ;
        per = ( m1 + m2 + m3 ) / 3 ;
        *( p + i )  = per ;
    }
    for ( i = 0 ; i < n ; i++ )
        printf ( "%d",  *( p + i ) ) ;
}
```

Memory Allocation

Array	malloc( )
Static	Dynamic
Compile Time	Execution Time

- Variables are created only during execution
- Static - Arrangement made at compilation time
  - Arrangement - Offset + Instruction to create var
- Dynamic - Arrangement made at execution time
  - Arrangement - Call to malloc( )

Allocation

```
int  x, y ;
main( )
{
    int  j ;  float  k ;  int  *p ;
    p = ( int * ) malloc ( 20 ) ;
}
```

Compiler

- Symbol table entries for variables
  - Name, Scope, Length
  - Offset from DS (global), SS (local)
- Instructions for local variables
- Call to malloc( )
- Creates OBJ and discards Symbol Table

Linker

- Our object code + Library's object code
- Create EXE file format (Header to help loader)

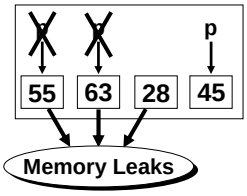
Loader

- If too many globals - Loader fails
- If too many locals - Stack overflow
- If too much dynamic demand - Heap full

Fatal

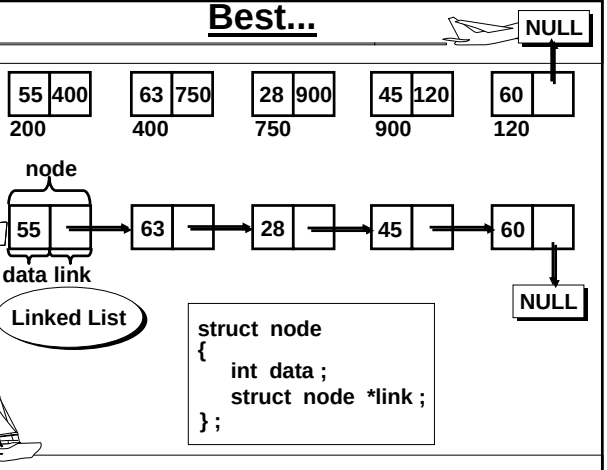
Better Still...

```
int  *p[  ] ;
char  ch = 'Y' ;
while ( ch = = 'Y' )
{
   scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
   per = ( m1 + m2 + m3 ) / 3 ;
   p[ i ] = ( int * ) malloc ( 2 ) ;
   *( p + i ) = per ;
   printf ( "Another student y/n" ) ;
   ch = getche( ) ;
```



Memory Leaks

Best...



Linked List

```
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
```

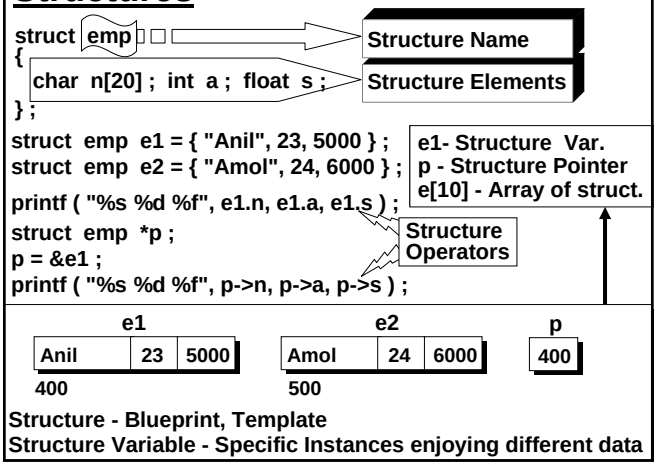
Procedural Programming

Yashavant Kanetkar

Objectives

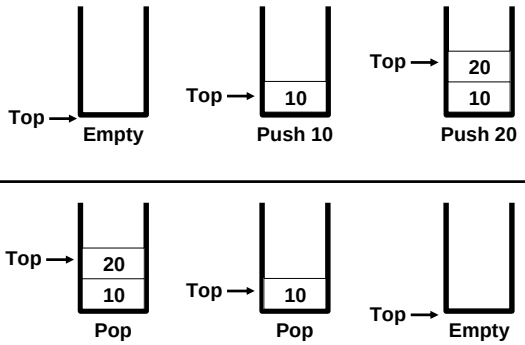
- Disadvantages of Procedural Programming
- The need for Object-Oriented Programming
- How structures in C++ are different

Structures



Structure - Blueprint, Template
Structure Variable - Specific Instances enjoying different data

Stack



Procedural Programming

```
Data1
Data2
Data3
::::::
Statement 1
Statement 2
Statement 3
Statement 4
Statement 5
Statement 6
:::::
```

```
void push ( ... )
{
   // code
}
void pop ( ... )
{
   // code
}
main( )
{
   Call  push
   Call  pop
}
```

- Functions
- Routines
- Procedures

- Sequence
- Decision
- Repetition
- Case

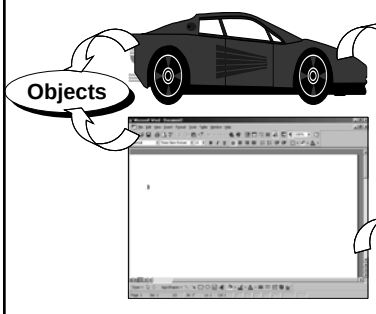
Control Inst.

Local variables

- AKA Structured Prog.
- Crux - Functional Abstrac^{n}
- Reuse at function level
- Ex. C, Pascal, Ada

Limitations:
- Poor real world model
- No importance to data
- No true reuse

Real-World Model

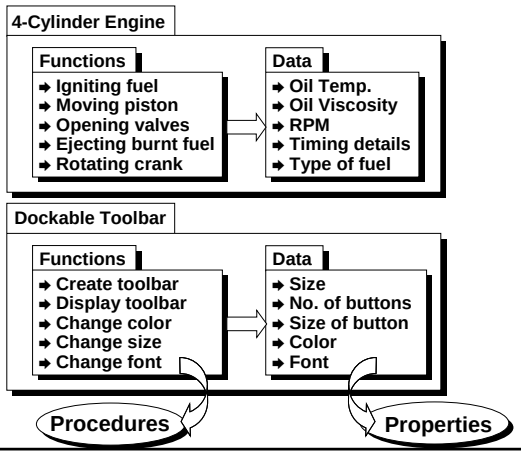


- 4-Cylinder Engine
- Synchromesh Gearbox
- Anti-lock Break
- 1.5 Ton AC
- Karaoke Sound

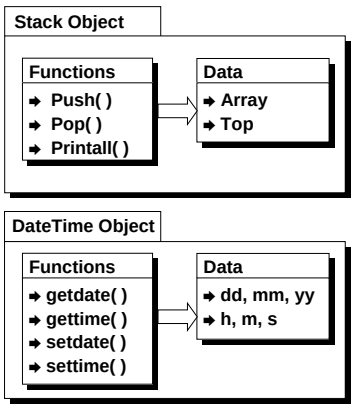
Reusable Objects

- Frame Window
- Dockable Toolbar
- Popup Menu
- Flat Scrollbar
- Push Button

Object Contents



More Objects



Object Oriented Solution

Yashavant Kanetkar

Objectives

- Basics of Classes and Objects
- Access modifiers
- Data Members and Member Functions

Procedural Solution

datetime.c

```
struct  datetime
{
    int  dd, mm, yy ;
    int  h, m, s ;
} ;
void getdate( )
{
    // code
}
void setdate ( ... )
{
    // code
}
```

stack.c

```
struct stack
{
    int  top ;
    int  a[10] ;
} ;
void push ( ... )
{
    // code
}
void pop ( ... )
{
    // code
}
```

Use

```
struct stack s1 ;
s1.a[4] = 8 ;
s1.top = -66 ;
```

Limitations

- No importance to data
- Stress on procedures

OO Solution

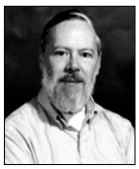
stack.c

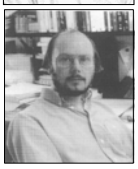
```
struct  stack
{
   private :
      int  top ;
      int  a[10] ;
   public :
      void push ( ... )
      {
         // code
      }
      void pop ( ... )
      {
         // code
      }
} ;
```

Use

```
stack  s1, s2 ;
s1.push ( .. ) ;
s1.push ( .. ) ;
s2.push ( .. ) ;
s2.push ( .. ) ;
s2.pop( ) ;
```

```
s1.top = -1 ;   ✖
s1.a[ 4 ] = 45 ; ✖
```





Facts

- Ex. C++, Java, SmallTalk
- 1980, AT&T Bell Labs
- Bjarne Stroustrup

Difference

C — Collection of Data

C++ — Collection of Data & Funs. — Classes

Class
- Blueprint, Template
- Data & Functions

Object
- Specific Data
- Functions

General — Unique

Class Stack	Object s1	Object s2
- a[10] - top	- 5 elements - top = 4	- 7 elements - top = 6
- push( ), pop( )	push( ), pop( )	push( ), pop( )

Classes In C++

Class members are by default private

```
# include <iostream.h>
class  a
{
   int  i ;
   float  j ;
} ;
void main( )
{
   a  z1 = { 10, 3.14 } ;
   cout  <<  z1.i  <<  z1.j ;
   a  z2 = { 20, 6.28 } ;
   cout  <<  z2.i  <<  z2.j ;
}
```

class is optional

z1

i	j
10	3.14

z2

i	j
20	6.28

z1, z2 - Objects

cout - Console Output
<< - Insertion Operator

Overloaded operator

```
cout  <<  z1.i  <<  z1.j ;   ✓
printf ( "%d %f", z1.i, z1.j ) ;
```

Making It Work

```
# include <iostream.h>
class  a
{
public :
      int  i ;  float  j ;
} ;

void main( )
{
   a  z = { 10, 3.14 } ;
   cout  <<  z.i   <<  z.j ;
}
```

Running C++ Programs

File Menu

New

Win32 Console Application

Supply Application name as 'Sample'

Select 'An Empty Project'

File Menu

New

C++ Source File

Supply File name

Type & Save

Build 'Sample.exe'

Execute 'Sample.exe'

!

Ctrl F5

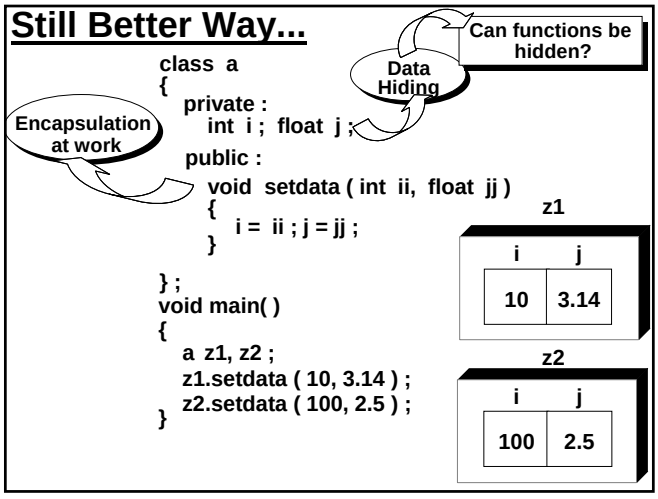
Classes In C++

Yashavant Kanetkar

Objectives

- How to work with multiple objects
- How to print the data
- How to construct objects

Still Better Way...



Which Is Better

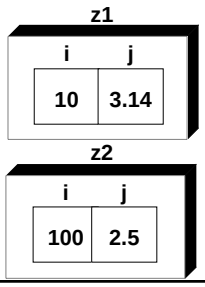
```
#include <iostream.h>
class sample
{
   public :
      int  age ;
} ;
void main( )
{
   sample  s1, s2 ; int  x ;
   cin  >>  x ;
   if ( x > 0 )
      s1.age = x ;
   cin >>  x ;
   if ( x > 0 )
      s2.age = x ;
}
```

```
#include <iostream.h>
class sample
{
   private :
      int  age ;
   public:
      void setdata ( int x )
      {
         if ( x > 0 )
            age = x ;
      }
} ;
void main( )
{
   sample  s1, s2 ; int  x ;
   cin >> x ;
   s1.setdata ( x ) ;
   cin >> x ;
   s2.setdata ( x ) ;
}
```

Printing The Data

```
#include  <iostream.h>
class  a
{
   private :
      int  i ;  float  j ;
   public :
      void  setdata ( int  ii,  float  jj )
      {
         i =  ii ; j = jj ;
      }
      void  printdata( )
      {
         cout  <<  i << "   "  <<  j  ;
      }
} ;
```

```
void main( )
{
   a  z1, z2 ;
   z1.setdata ( 10, 3.14 ) ;
   z2.setdata ( 100, 2.5 ) ;
   z1.printdata( ) ;
   z2.printdata( ) ;
}
```



this Pointer

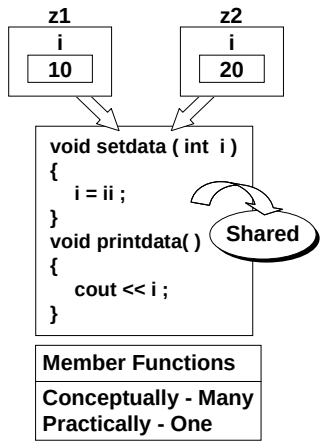
Yashavant Kanetkar

Objectives

- What comprises an object
- What is a this pointer
- Utility of the this pointer

```
class a
{
   private :
      int i ;
   public :
      void setdata ( int  ii )
      {
         i = ii ;
      }
      void printdata( )
      {
         cout << i ;
      }
} ;
void main( )
{
   a  z1, z2 ;
   z1.setdata ( 10 ) ;
   z2.setdata ( 20 ) ;
}
```

How Many Copies



Member Functions
Conceptually - Many Practically - One

The *this* Pointer

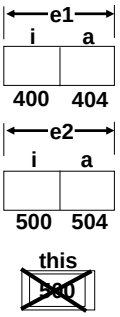
```
class  ex
{
    private :
        int  i ;  float  a ;
    public :
        void setdata ( int  ii, float  aa )
        {
            this -> i = ii ;
            this ->a = aa ;
        }
} ;
void main( )
{
   ex  e1, e2 ;
   e1.setdata ( 5, 5.5 ) ;
   e2.setdata ( 10, 10.5 ) ;
}
```

Optional

Caution!! this Pointer cannot be modified

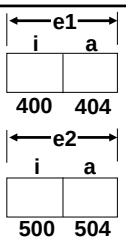
```
void setdata ( ex* const this,
               int  ii, float  aa )
```

```
ex::setdata ( &e1, 5, 5.5 ) ;
ex::setdata ( &e2 10, 10.5 ) ;
```



Is *this* Necessary

```
class  ex
{
    private :
        int  i ;  float  a ;
    public :
        void setdata ( int  i, float  a )
        {
            this -> i = i ;
            this ->a = a ;
        }
} ;
void main( )
{
   ex  e1, e2 ;
   e1.setdata ( 5, 5.5 ) ;
   e2.setdata ( 10, 10.5 ) ;
}
```



Access Specifier & Constructor

Yashavant Kanetkar

Objectives

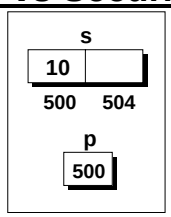
- Hiding versus security
- How constructors work
- Advantages of constructors
- Overloaded Constructors

Recap...

- Classes - Datatypes
- Objects - Variables
- Access Specifiers - private, public
  - Facilitate Hiding
- Why create a project
  - To compile all files at one shot
- Why create Win32 Console application
  - Output in DOS window
  - cout - sends output to console

Hiding vs Security

```
class sample
{
   private :
      int i ; float j ;
   public:
      void display( )
      {
         cout << i ;
      }
} ;
void main( )
{
   sample s ;
   int *p ;
   p = ( int * ) &s ;
   *p = 10 ;
}
```



"The protection of private data can be circumvented through pointers. But this, of course, is cheating.
C++ protects against accident rather than deliberate fraud." - ***Bjarne Stroustrup***

C vs C++

```
struct stack
{
   int top ; int a[ 10 ] ;
} ;
void init ( struct stack *s )
{
   s -> top = 0 ;
}
void push ( struct stack *s, int x )
{
   s -> a[s -> top] = x ; s -> top++ ;
}
void main( )
{
   stack s1, s2 ;
   init ( &s1 ) ; push ( &s1, 10 ) ;
   init ( &s2 ) ; push ( &s2, 20 ) ;
}
```

Advantages

- Cleaner
- Better organized
- Sure initialisation

```
class stack
{
   private :
      int top ; int a[ 10 ] ;
   public :
      stack( )
      {
         top = 0 ;
      }
      void push ( int x )
      {
         a[ top ] = x ;
         top++ ;
      }
} ;
void main( )
{
   stack s1 ;
   s1.push ( 10 ) ;
   stack s2 ;
   s2.push ( 5 ) ;
}
```

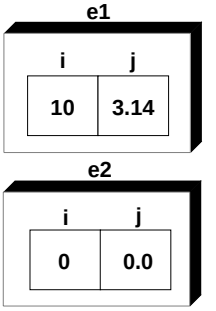
Constructors

```
class ex
{
   private :
      int i ; float j ;
   public :
      ex ( int ii, float jj )
      {
         i = ii ; j = jj ;
      }
      ex( )
      {
         i = 0 ; j = 0.0 ;
      }
} ;
void main( )
{
   ex e1 ( 10, 3.14 ) ;
   ex e2 ;
}
```

2 Argument Constructor

Overloaded Functions

0 Argument Constructor



Either I do it or you do it

Object Creation

- Allocate space in memory
- Call the constructor

Function Overloading

Yashavant Kanetkar

Objectives

- Function overloading
- Default values for arguments
- Pitfalls in overloading functions

Function Overloading

```
void  set ( int  i )
{
}
void  set ( int  i, int  j )
{
}
void  set ( int  i,  float  j )
{
}
void  set ( float  jj, int  ii )
{
}   ✖
int  set ( float  jj, int  ii )
{
}   ✖
```

```
void main( )
{
   set ( 10 ) ;
   set ( 10, 3 ) ;
   set ( 10, 3.14 ) ;
   set ( 3.14, 10 ) ; ✖
   set ( 3.14, 10 ) ; ✖
}
```

Arguments must differ in
- Number
- Order
- Type

Different return types not enough for overloading

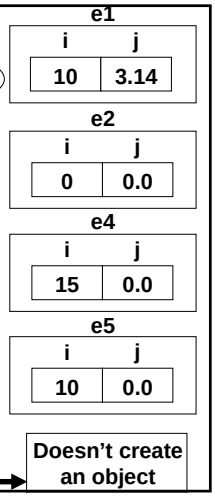
Two In One

```
class  ex
{
   private :
       int  i ;  float  j ;
   public :
       ex ( int  ii = 0,  float  jj = 0.0 )
       {
            i =  ii ;  j = jj ;
       }
} ;
void main( )
{
   ex  e1 ( 10, 3.14 ) ;
   ex  e2 ;
   ex  e3 ( , 1.2 ) ;
}
```

Can we write 0-Arg Ctor ✖

Error-prone for too many args ✖

What if	
ex e4 (15) ; ✓	ex e6 = 2, 1.1 ; ✖
ex e5 = 10 ; ✓	ex e7( ) ; ✓ → Doesn't create an object



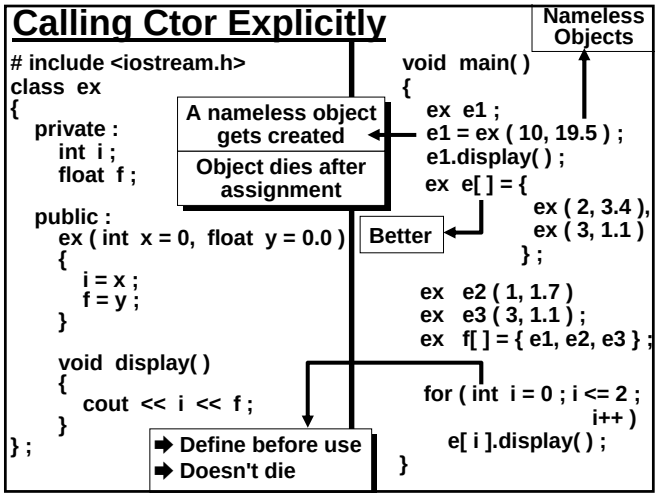
Constructors

Yashavant Kanetkar

Objectives

- Array of objects
- Calling constructors explicitly
- Pointer to an Object

Calling Ctor Explicitly



Pointer To An Obj.

```
# include <iostream.h>
class  ex
{
   private :
      int  i ;  float  f ;
   public :
      ex ( int  x,  float  y )
      {
         i = x ;  f = y ;
      }
      void  set ( int  x,  float  y )
      {
         i = x ;  f = y ;
      }
      void  display( )
      {
         cout  <<  i  <<  f ;
      }
} ;
```

```
void  main( )
{
   ex  e ( 1, 2.5 ) ;
   e.display( ) ;
   e.set ( 2, 5.5 ) ;
   e.display( ) ;
   fun ( &e ) ;
   e.display( ) ;
}
void  fun ( ex  *p )
{
   p -> set ( 3, 8.5 ) ;
}
```

1 2.5

2 5.5

3 8.5

this ptr = address of e

Set new data - Call set( )
Set data from other fun. - Pass addr.

Types of Ctors

Yashavant Kanetkar

Objectives

- Normal Constructor
- Copy constructor
- Overloaded Assignment Operator
- Which gets called when

Readymades

```
class  ex
{
} ;
```

- 0-Arg constructor
- Copy constructor
- Overloaded =

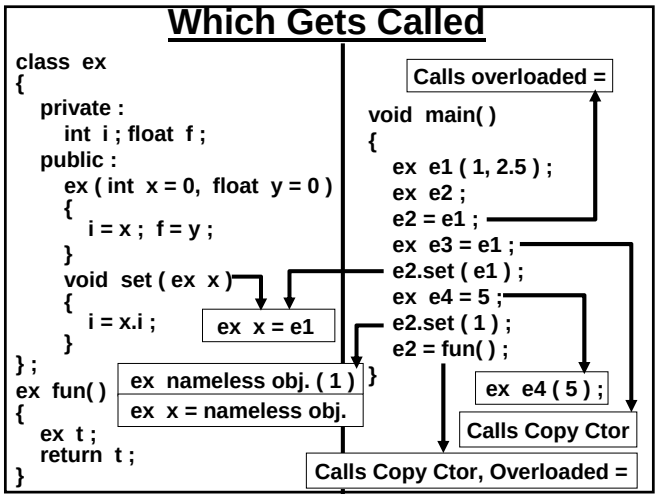
```
void  main( )
{
   ex  e1 ;
   ex  e2 ;
   e2 = e1 ;
   ex e3 = e2 ;
}
```

0-Arg

Overloaded =

Copy Constr.

Which Gets Called



Operator Overloading

Yashavant Kanetkar

Objectives

- What is operator overloading
- Need for Operator overloading
- Which operators cannot be overloaded

Operator Overloading

```
# include <iostream.h>
class  comp
{
  private :
    double  r, i ;
  public :
    comp ( double rr = 0,
           double ii = 0)
    {
      r = rr ;
      i = ii ;
    }
    void print( )
    {
      cout  <<  r  <<  i ;
    }
```

Return Type

```
comp operator +  ( comp c2 )
{
  comp  t ;
  t.r = r + c2.r ;
  t.i = i + c2.i ;
  return  t ;
}
} ;

void main( )
{
  comp  a ( 1.0, 1.0 ) ;
  comp  b ( 2.0, 2.0 ) ;
  comp  c ;
  c = a + b ;
  cout << "c = " ;
  c.print( ) ;
}
```

c = a.operator + (b) ;

c = nameless obj.

Tips About Overloading

- The operators . , :: , ? and : cannot be overloaded
- Precedence cannot be changed through overloading
- Overloading allowed for user-defined types

a.b - Can't be broken Clumsy	:: - Resolves scope ? : - Don't manipulate

Pre, Post and References

Yashavant Kanetkar

Objectives

- Overloading of pre & post incr. operators
- Need of References
- Subtleties of References

Pre & Post

```
class  index
{
   private : int  count ;
   public :
      index( )
      {
         count = 0 ;
      }
      void display( )
      {
         cout << count ;
      }
      index operator ++ ( )
      {
         index  t ;
         count++ ;
         t.count = count ;
         return  t ;
      }
      index operator ++ ( int  n )
      {
         index  t ;
         t.count = count ;
         count++ ;
         return  t ;
      }
} ;
void main( )
{
   index  i, j, k ;
   j = ++i ;
   k = i++ ;
   i.display( ) ;
   j.display( ) ;
   k.display( ) ;
}
```

i.operator ++( )

i.operator ++(0)

2 1 1

References

```
int  i = 10 ;
int  &j = i ;
cout  <<  i  <<  j ;
```

10 10

j - Reference
i - Referrent

```
j = 20 ;
cout  <<  i  <<  j ;
```

20 20

i	j
10	300
300	400

```
i = 30 ;
cout  <<  i  <<  j ;
```

30 30

- ➡ Changing a reference changes referrent
- ➡ A reference is a const pointer
- ➡ A reference is automatically de-referenced
- ➡ Hence when we use a reference we reach a referrent

Subtleties

```
int  i = 10 ;
int  &j ;
j = i ;
```

✖

Reference must always be initialised

```
int  i = 10, k = 20 ;
int  &j = i ;
j = k ;
k = 30 ;
cout << i << j << k ;
```

20 20 30

Once tied always remains tied

- ➡ Multiple references are allowed
- ➡ No way to create a reference to a reference
- ➡ Array of references is not allowed

More About References

Yashavant Kanetkar

Objectives

- Call by value
- Call by address
- Call by reference
- Which call to use when

Different Calls

```
# include <iostream.h>
struct  data
{
   int  i ;  float  f ;
} ;
class  emp
{
  public :
     void fun1 ( data  x )
     {
        x.i = 2 ;  x.f = 5.5 ;
     }
     void fun2 ( data  *y )
     {
        y->i = 2 ;  y->f = 5.5 ;
     }
```

Adv
- Automatic de-referencing
- Avoids a copy

```
     void fun3 ( data &z )
     {
        z.i = 3 ;  z.f = 10.5 ;
     }
} ;
void main( )
{
     data  d = { 1, 2.2 } ;
     emp  e ;
     e.fun1 ( d ) ;
     cout  <<  d.i  <<  d.f ;
     e.fun2 ( &d ) ;
     cout  <<  d.i  <<  d.f ;
     e.fun3 ( d ) ;
     cout  <<  d.i  <<  d.f ;
}
```

Can all be fun( )

1 2.2

2 5.5

3 10.5

Are References Necessary

```
class  comp
{
  public :
     comp operator + ( comp  c2 )
     {
         comp  t ;
         t.r = r + c2.r ;
         t.i = i + c2.i ;
         return  t ;
     }
} ;
void  main( )
{
    comp  a ( 1.0, 1.0 ) ;
    comp  b ( 2.0, 2.0 ) ;
    comp  c ;
    c = a + b ;
}
```

How to avoid a copy?

✓ Use Reference

Use Pointer ✓

c = a.operator + (b) ;

Dynamic Memory Allocation - I

Yashavant Kanetkar

Objectives

- What is Static Memory Allocation
- What is Dynamic Memory Allocation
- How memory is allocated on stack and heap
- Differences between static & dynamic allocation

Dynamic Memory Allocation

```
int  i ;
float  a ;
struct  emp
{
    char  n[ 20 ] ;
    int  a ;
    float  s ;
} ;
emp  e ;
```

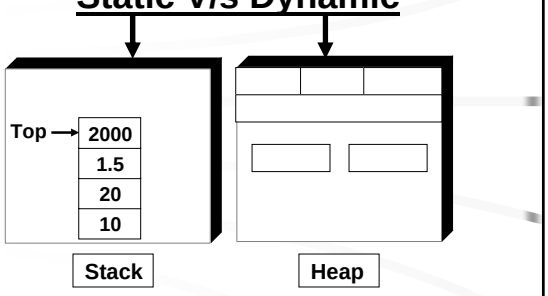
```
int  *i ;  float  *a ;
i = ( int * ) malloc ( sizeof ( int ) ) ;
a = ( float * ) malloc ( sizeof ( float ) ) ;
struct  emp
{
    char  n[ 20 ] ;
    int  a ;
    float  s ;
} ;
emp  *e ;
e = ( emp * ) malloc ( sizeof ( emp ) ) ;
```

```
i = new  int  ;
a = new  float ;
e = new  emp ;
```

new

- Operator
- Does DMA

Static V/s Dynamic



Allocation

```
int  x, y ;
void main( )
{
     int  j ;  float  k ;  int  *p ;
     p = new  int ;
}
```

Compiler

- Symbol table entries for variables
  - Name, Scope, Length
  - Offset from DS (global), SS (local)
- Emits instructions for local variables
- Emits call to new( )
- Creates OBJ code and discards Symbol Table

Linker

- Our object code + Library's object code
- Create EXE file format (Header to help loader)

Loader

- If too many globals - Loader fails
- If too many locals - Stack overflow

Fatal

- If too much dynamic demand - Heap full

Dynamic Memory Allocation - II

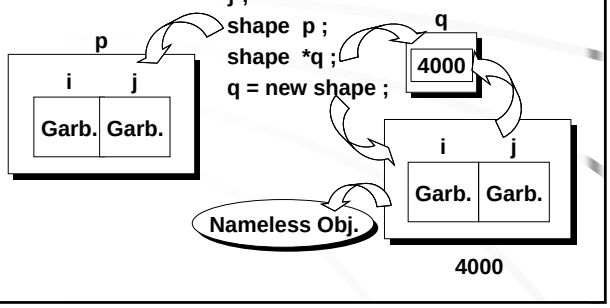
Yashavant Kanetkar

Objectives

- Difference between new and malloc( )
- Difference between delete and free( )
- How to avoid memory leaks & dangling ptrs
- Allocating memory dynamically for an array

Named & Nameless Objects

```
class  shape
{
   private :
      int  i, j ;
} ;
shape  p ;
shape  *q ;
q = new shape ;
```



Are *new* And *malloc( )* Same

```
class  ex
{
    private :
        int  i ;  float  a ;
    public :
        ex( )
        {
            i = 0 ;
            a = 0.0 ;
        }
        ex ( int  ii, float  aa )
        {
            i = ii ;
            a = aa ;
        }
} ;
```

```
void main( )
{
    ex  *p1, *p2 ;
    p1 = new  ex ;
    p2 = new  ex ( 10, 3.5 ) ;
    delete  p1 ;
    delete  p2 ;
}
```

operator

- new allocates memory, calls constructor
- What is allocated must be de-allocated
- delete calls destructor, deallocates memory

Avoid Memory Leaks - I

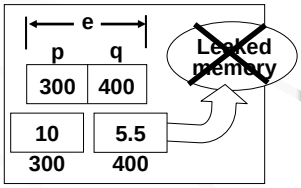
```
class  ex
{
    private :
        int  *p ;  float  *q ;
    public :
        ex ( int  ii, float  aa )
        {
            p = new  int ;
            q = new  float ;
            *p = ii ;
            *q = aa ;
        }
        ~ex( )
        {
            delete  p ;
            delete  q ;
        }
} ;
```

Destructor

Doesn't delete q

```
void main( )
{
    void  f( ) ;
    f( ) ;
}
void f( )
{
    ex  e ( 10, 5.5 ) ;
}
```

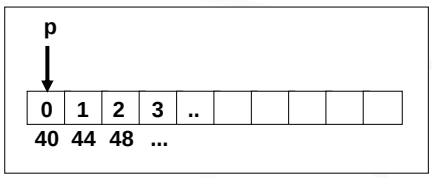
If new is used in constructor, use delete in destructor



Array Allocation

```
int   *q ;
q = new  int ;

int  *p ;
p = new  int[ 10 ] ;
for ( int  i = 0 ; i  < 10 ; i++ )
    * ( p + i ) = i ;
```

Can be a variable

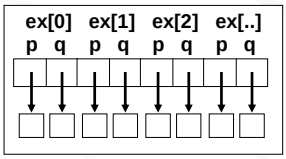


Avoid Memory Leaks - II

```
class  ex
{
    private :
       int  *p ;  float  *q ;
    public :
       ex ( )
       {
            p = new  int ;
            q = new  float ;
            *p = 0 ;
            *q = 0 ;
       }
       ~ex( )
       {
          delete  p ;
          delete  q ;
       }
} ;
```

```
void  main( )
{
    void  f( ) ;
    f( ) ;
}
void  f( )
{
   ex  *z ;
   z =  new ex[ 10 ] ;
   delete  z ;
}
```

delete [ ] z ;



Static Members

Yashavant Kanetkar

Objectives

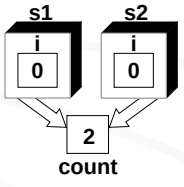
- Static Data Members
- Static Member Functions
- Static Storage Class
- Comparison of Instance, Static and Friend Fun.

Static

```
# include <iostream.h>
class  sample
{
      int  i ;
      static  int  count ;
    public :
      sample( )
      {
          i = 0 ;
          count ++ ;
      }
      static void objects( )
      {
          cout << count ;
      }
} ;
int  sample::count = 0 ;
```

```
void main( )
{
    sample  s1, s2 ;
    sample :: objects( ) ;
}
```

s1.objects( ) ✓
s2.objects( ) ✓



static functions can access only static data

Difference

Function Type	Access Private Data / Functions	Within class scope	Invoked using Object
Member	✓	✓	✓
Static	✓ only static	✓	✗
Friend	✓	✗	✗

Reuse Mechanisms

Yashavant Kanetkar

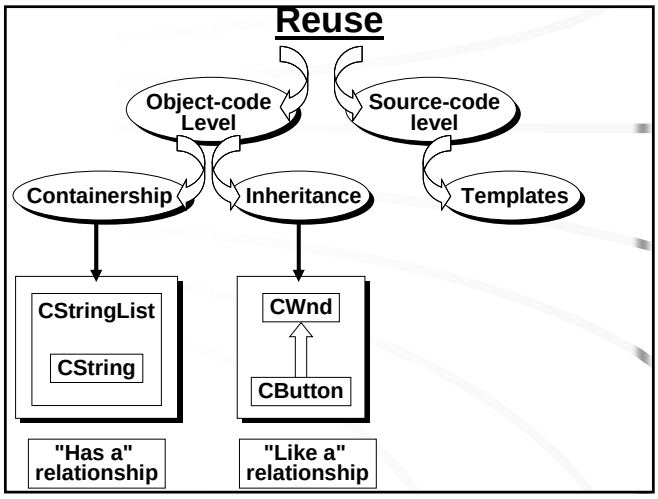
Objectives

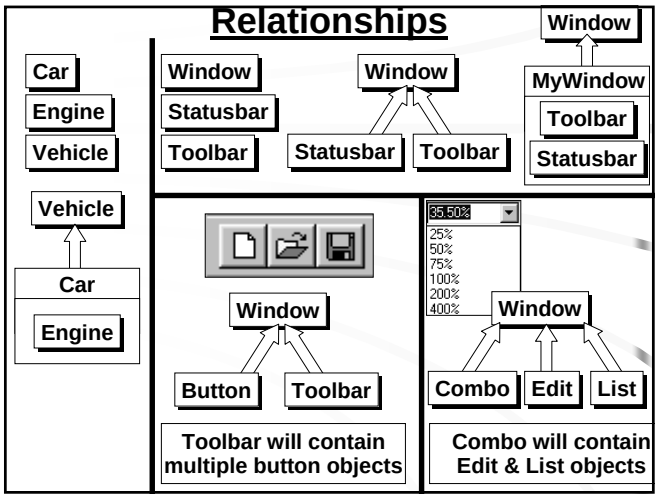
- Ways to reuse existing code
- Inheritance reuse mechanism
- Containership reuse mechanism
- What are templates
- When to use which mechanism

What Next

Class

- Data Hiding
- Guaranteed initialization of data
- Implicit type conversion – double to comp
- Function overloading
- Operator overloading
- User controlled memory management
- Shared data and functions


Reuse
Object-code Level
Source-code level
Containership
Inheritance
Templates
CStringList
CString
CWnd
CButton
"Has a" relationship
"Like a" relationship


Relationships
Car
Engine
Vehicle
Vehicle
Car
Engine
Window
Statusbar
Toolbar
Window
Statusbar
Toolbar
Window
MyWindow
Toolbar
Statusbar
Window
Button
Toolbar
Toolbar will contain multiple button objects
35.50%
25%
50%
75%
100%
200%
400%
Window
Combo
Edit
List
Combo will contain Edit & List objects

Containership and Inheritance

Yashavant Kanetkar

Objectives

- How containership works
- How Inheritance works
- Sample Programs

Containership

```
class  string
{
   private :
      char  str[ 100 ] ;
   public :
      string ( char  *s = " " )
      {
      }
      void  print( )
      {
      }

      // functions
} ;
```

```
class  stringlist
{
   private :
      string  s[ 50 ] ;
      int  c ;
   public :
      stringlist( )
      {
         c = 0 ;
      }
      void  add ( string  t )
      {
         s[ c ] = t ;  c++ ;
      }
      void  printall( )
      {
         // code
      }
} ;
```

Inheritance

```
# include <iostream.h>
class  index
{
   private :
      int  count ;
   public :
      index( )
      {
         count = 0 ;
      }
      void display( )
      {
         cout << count ;
      }
      void operator ++ ( )
      {
         count ++
      }
} ;
```

protected :

i.operator ++( ) ;

```
class  index1: public index
{
   public :
      void operator - -( )
      {
         count - - ;
      }
} ;
void main( )
{
   index  i ;
   i.display( ) ;
   i++ ;
   i.display( ) ;
   i- - ;
   i.display( ) ;
}
```

index1 i ;

0

1

0

Virtual Functions

Yashavant Kanetkar

Objectives

- How to achieve runtime polymorphism
- How runtime polymorphism works
- Purpose of pure virtual functions

```
#include <iostream.h>
class  shape
{
   public :
      virtual void draw( )
      {
         cout << "shape" ;
      }
} ;

class  circle : public  shape
{
   public :
      void draw( )
      {
         cout << "circle" ;
      }
} ;
```

Keyword

```
class rectangle : public shape
{
   public :
      void draw( )
      {
         cout << "rect" ;
      }
} ;
void main( )
{
   shape  *p ;
   circle  c ;
   rectangle  r ;
   p = &c ;  // no error
   p -> draw( ) ;
   p = &r ;
   p -> draw( ) ;
}
```

Same effect in data?

No. Data isn't virtual

~~shape~~ circle

~~shape~~ rect

Pure Virtual Functions

```
#include <iostream.h>
class  shape
{
   public :
      virtual void draw( ) = 0 ;
} ;
class  circle : public  shape
{
   public :
      void draw( )
      {
         cout << "circle" ;
      }
} ;
```

- Same Interface
- Different Implementation

```
class  rectangle : public  shape
{
   public :
      void draw( )
      {
         cout << "rectangle" ;
      }
} ;
void main( )
{
   circle  c1, c2, c3, c4, c5 ;
   rectangle  r1, r2, r3, r4, r5 ;
   shape  *p[10] = { &c1, &r4, .. } ;
   for ( int  i = 0 ; i <= 9 ; i++ )
      p[ i ] -> draw( ) ;
}
```

Summary...

Upcasted pointer - base *b ;
derived d ;
b = &d ;

Abstract class - Contains at least one pure virtual function
- May contain non-virtual function
- Object creation not possible

Virtual fun. mechanism - Upcasted pointer
- If function names are same
  - Base class version if non-virtual
  - Derived class version if virtual

A virtual function is early bound if called through object

A virtual function is late bound if called through pointer (upcasted or not)

Linked List

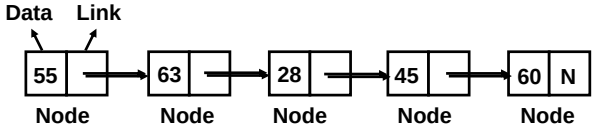
Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- Limitation of Arrays
- What are linked lists
- How to create linked lists
- How to create a linked list of records

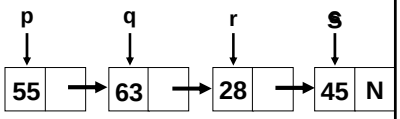
Why Linked Lists

- Array size cannot be changed
- Adjacent locations may not be available
- Insertion / Deletion is tedious
- Solution – Linked List



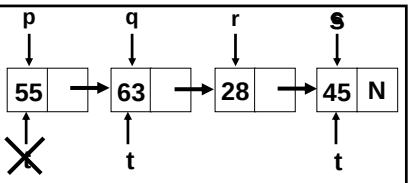
- No size limit
- Operations are easy

Linked List




```
# include <iostream>
using namespace std ;
struct  node
{
    int  data ;
    node  *link ;
} ;
int main( )
{
   node *p, *q, *r, *s ;
   p = new node ;
   q = new node ;
   r = new node ;
   s = new node ;
   p -> data = 55 ; q -> data = 63 ; r -> data = 28 ; s -> data = 45 ;
   p -> link =  q ;  q -> link = r ;  r -> link = s ;  s -> link = NULL ;
```

..Cont



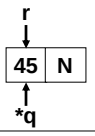

```
int main( )
{
   ...
   ...
   ...
   ...
   cout << p -> data << endl << q -> data  <<  endl  ;
   cout << r -> data << endl <<  s -> data  <<  endl  ;
   cout << p -> link -> data << endl ;
   cout << p -> link -> link -> data << endl ;
   t = p ;
   while ( t != NULL )
   {
       cout << t -> data << endl  ;   t = t -> link ;
   }
}
```

Most General

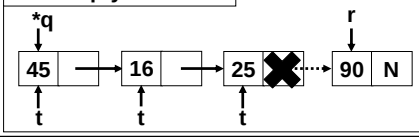
```
# include <iostream>
using namespace std ;
    struct  node
    {
        int  per ;  node  *link ;
    } ;
    void  add ( node **q, int  pp ) ;
    int  main( )
    {
        node  *p ;  char  ch = 'Y' ;  int  pp, m1, m2, m3 ;
        p = NULL ;
        while ( ch == 'Y' )
        {
            cin >> m1 >> m2 >> m3 ;
            pp = ( m1 + m2 + m3 ) / 3 ;
            add ( &p, pp ) ;
            cout <<  "Another student Y/N" ;
            cin >> ch ;
        }
    }
```

```
void  add ( node **q, int  pp )
{
    node  *r ;   node  *t ;
    r = new node ;
    r  -> per = pp ;   r -> link = NULL ;
    if ( *q == NULL )
       *q = r ;
    else
    {
       t = *q ;
       while ( t -> link != NULL )
             t = t -> link ;
       t -> link = r ;
    }
}
```

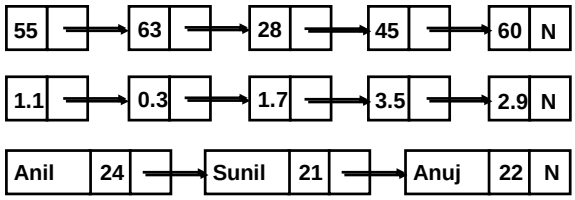
Empty linked list



Non-empty linked list



Variety...




```
struct  node
{
    char name[ 20 ] ;
    int age ;
    node  *link ;
} ;
```

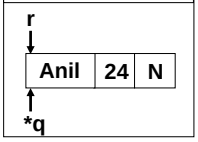
LL Of Records

```
# include <iostream>
using namespace std ;
struct  node
{
    char  name [ 20 ] ;  int  age ;  node  *link ;
} ;
void  add ( struct node **q, struct  node  n ) ;
int  main( )
{
    node  *p ;
    node  t ;   char  ch = 'Y' ;
    p = NULL ;
    while ( ch == 'Y' )
    {
        cout << "\nEnter name & age: " ;
        cin >> t.name >> t.age ;
        add (  &p, t ) ;
        cout << "Another student Y/N" ;
        cin >> ch ;
    }
}
```

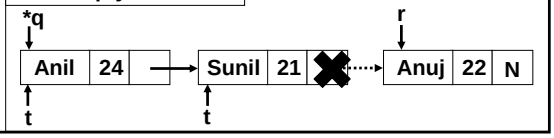
Contd...

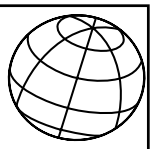
```
void  add ( node **q,  node  n )
{
    node  *r ;  node  *t ;
    r = new  node ;
    *r = n ;  r -> link = NULL ;
    if ( *q == NULL )
       *q = r ;
    else
    {
       t = *q ;
       while ( t -> link != NULL )
            t = t -> link ;
       t -> link = r ;
    }
}
```

Empty linked list



Non-empty linked list





Operations on Linked List

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- How to append a new node to a link list
- How to add a new node at the beginning of a LL
- How to insert a new node in a linked list
- How to delete a node from a linked list
- How to delete all nodes in the list

LL Operations

```
# include <iostream>
using namespace std ;

int main( )
{
    linkedlist l ;
    int  c ;
    l.append ( 14 ) ;
    l.append ( 30 ) ;
    l.append ( 25 ) ;
    l.append ( 17 ) ;
    l.display( ) ;

    l.addatbeg ( 7 ) ;
    l.addatbeg ( 58 ) ;
    l.display( ) ;

    l.insert ( 7, 0 ) ;
    l.insert ( 2, 1 ) ;
    l.insert ( 5, 99 ) ;
    l.display( ) ;
    c = l.count( ) ;
    cout  <<  c ;
    l.del ( 30 ) ;
    l.del ( 10 ) ;
    l.display( ) ;
    c = l.count( ) ;
    cout << c ;

    l.deleteall( ) ;
}
```

linkedlist Class

```
class linkedlist
{
   private :
      struct node
      {
         int  data ;  node  *link ;
      } *p ;
   public :
      linkedlist( ) ;
      ~linkedlist( ) ;
      void append ( int num ) ;
      void addatbeg ( int num ) ;
      void insert ( int loc, int num ) ;
      void display( ) ;
      int count( ) ;
      void del ( int num ) ;
      void deleteall( ) ;
} ;
```

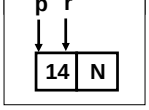
Ctor & Dtor

```
linkedlist :: linkedlist( )
{
   p = NULL ;
}

linkedlist :: ~linkedlist( )
{
   deleteall( ) ;
}
```

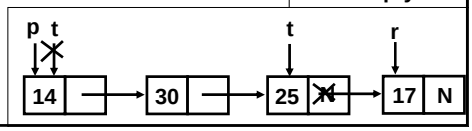
Append

```
void linkedlist :: append ( int  num )
{
   node  *r, *t ;
   r = new node ;
   r -> data = num ;
   r -> link = NULL ;
   if ( p == NULL )
      p = r ;
   else
   {
      t = p ;
      while ( t -> link != NULL )
         t = t -> link ;
      t -> link = r ;
   }
}
```

Empty List

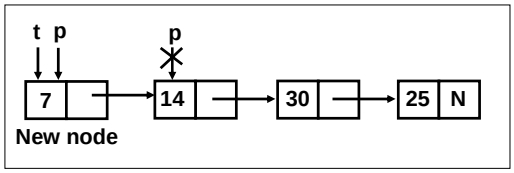


Non-Empty List



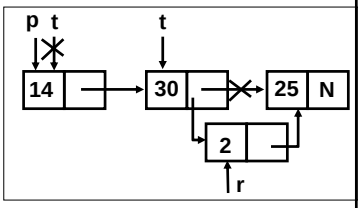
Add At Beginning

```
void linkedlist :: addatbeg ( int  num )
{
      node  *t ;
      t = new node ;
      t -> data = num ;
      t -> link = p ;
      p = t ;
}
```

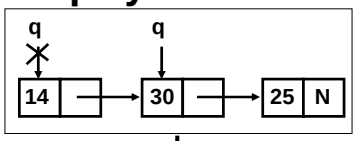


Insert Node

```
void linkedlist :: insert ( int  pos, int  num )
{
    node  *t , *r ;
    int  i ;
    t = p ;
    for ( i = 0 ; i < pos ; i++ )
    {
        if ( t -> link == NULL )
            break ;
        t = t -> link ;
    }
    r = new node ;
    r -> data = num ;
    r -> link = t -> link ;
    t -> link = r ;
}
```



Display And Count

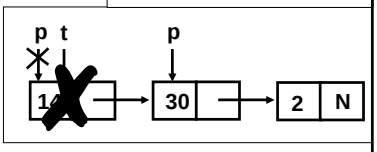



```
void linkedlist :: display( )
{
    node *q = p ;
    while ( q != NULL )
    {
       cout << q->data << endl ;
       q = q -> link ;
    }
}
```

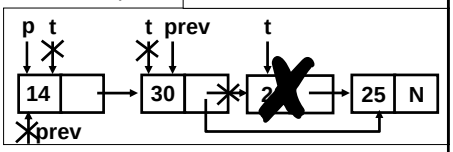
```
int linkedlist :: count( )
{
    node *q = p ;
    int c = 0 ;
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }
    return c ;
}
```

```
void linkedlist :: del ( int  num )
{
   node  *t, *prev ;
   t = p ;
   while ( t != NULL )
   {
      if ( t -> data == n )
      {
         if ( t == p )
            p = t -> link ;
         else
            prev -> link = t -> link ;
         delete t ;
         return ;
      }
      prev = t ;
      t = t -> link ;
   }
   cout << "\nEle. not found." ;
}
```

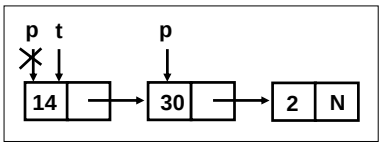
Node to be Deleted = 14



Node to be Deleted = 2



Delete All Nodes




```
void linkedlist :: deleteall( )
{
    node  *t ;
    while ( p != NULL )
    {
        t = p ;
        p = p -> link ;
        delete  t ;
    }
}
```

Ascending Order Linked List

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- How to create an ascending order linked list
- How to sort elements in the linked list

Ascending Order LL

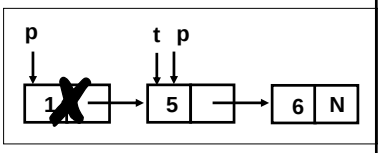
```
# include <iostream>
using namespace std ;

int main( )
{
    sortedll l ;
    int  c ;

    l.add ( 5 ) ;
    l.add ( 1 ) ;
    l.add ( 6 ) ;
    l.add ( 4 ) ;
    l.add ( 7 ) ;

    l.display( ) ;
    c = count ( p ) ;
    cout << "No. of elements =" << c ;
}
```

Sorted Linked List Class

```
class sortedll
{
   private :
      struct node
      {
         int   data ;  node  *link ;
      } *p ;

   public :
      sortedll( ) ;
      ~sortedll( ) ;
      void add ( int num ) ;
      void display( ) ;
      int count( ) ;
} ;
```

Ctor And Dtor

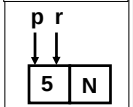
```
sortedll :: sortedll( )
{
    p = NULL ;
}

sortedll :: ~sortedll( )
{
   node *t ;
   while ( p != NULL )
   {
        t = p -> link ;
        delete p ;
        p = t  ;
   }
}
```



add( ) Function

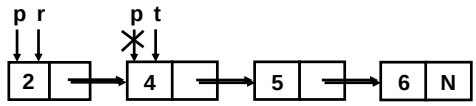
```
void sortedll :: add ( int  n )
{
   node  *r, *t ;
   t = p ;
   r = new node ;
   r -> data = n ;
   if ( p == NULL   || p -> data > n )
   {
      p = r ;
      p -> link = t ;
   }
```

Empty LL



New node

Addition At Beginning

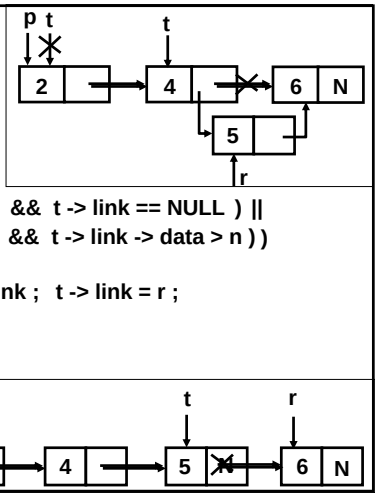


New node

Cont…

Cont…

```
    else
    {
        while ( t != NULL )
        {
            if ( ( t -> data <= n  &&  t -> link == NULL ) ||
                ( t -> data <= n  &&  t -> link -> data > n ) )
            {
                r -> link = t -> link ;   t -> link = r ;
                return ;
            }
            t = t -> link ;
        }
    }
}
```



Sorting

```
# include <iostream>
using namespace std ;

int main( )
{
    linkedlist l ;
    l.append (17 ) ;
    l.append ( 6 ) ;
    l.append ( 13 ) ;
    l.append ( 12 ) ;
    l.append ( 2 ) ;
    l.display( ) ;
    l.selectionsort( ) ;
    l.display( ) ;
}
```

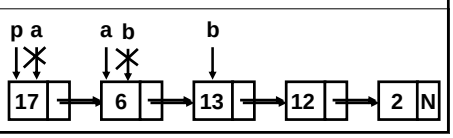
Sorted Linked List Class

```
class linkedlist
{
   private :
      struct node
      {
         int   data ;  node  *link ;
      } *p ;

   public :
      linkedlist( ) ;
      ~linkedlist( ) ;
      void append ( int num ) ;
      void display( ) ;
      void selectionsort( ) ;
      int count( ) ;
} ;
```

Ctor And Dtor

```
linkedlist :: linkedlist( )
{
    p = NULL ;
}

linkedlist :: ~linkedlist( )
{
   node *q ;
   while ( p != NULL )
   {
        q = p -> link ;
        delete p ;
        p = q  ;
   }
}
```

```
void linkedlist :: selectionsort( )
{
   node *a , *b ;
   int  n, i,  j, t ;
    a = p ;   n = count( ) ;
   for ( i = 0 ; i < n - 1 ; i++ )
   {
      b = a -> link ;
      for ( j = i + 1 ; j < n ; j++ )
      {
         if ( a -> data > b -> data )
         {
            t = a -> data ;   a -> data = b -> data ;   b -> data = t ;
         }
         b = b -> link ;
      }
      a = a -> link ;
   }
}
```



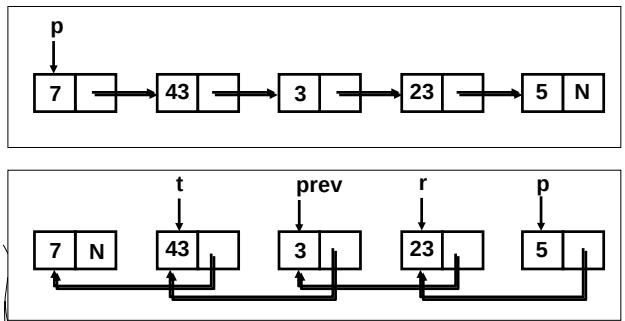
Reversing & Merging Linked Lists

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

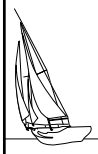
- Perform advanced operations on linked lists
- How to reverse a linked list
- How to merge two linked lists

Reversing LL



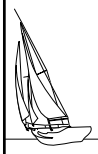
Program

```
# include <iostream>
using namespace std ;

int main( )
{
    linkedlist  l ;
    l.append ( 7 ) ;
    l.append ( 43 ) ;
    l.append ( 3 ) ;
    l.append ( 23 ) ;
    l.append ( 5 ) ;

    l.display( ) ;
    l.reverse( ) ;
    l.display( ) ;
}
```



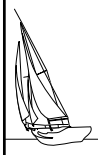
Linked List Class

```
class linkedlist
{
   private :
      struct node
      {
         int   data ;  node  *link ;
      } *p ;

   public :
      linkedlist( ) ;
      ~linkedlist( ) ;
      void append ( int num ) ;
      void reverse( ) ;
      void display( ) ;
} ;
```

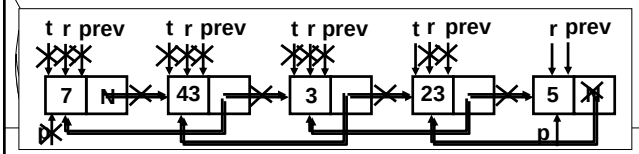


Ctor And Dtor

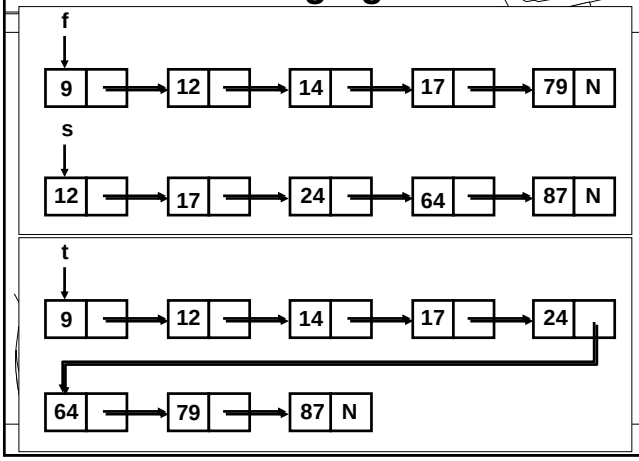
```
linkedlist :: linkedlist( )
{
    p = NULL ;
}

linkedlist :: ~linkedlist( )
{
   node *q ;
   while ( p != NULL )
   {
        q = p -> link ;
        delete p ;
        p = q  ;
   }
}
```



```
void linkedlist :: reverse( )
{
    node  *r, *prev, *t ;
    r = p ;    prev = NULL ;
    while ( r != NULL )
    {
        t = prev ;
        prev = r ;
        r = r -> link ;
        prev -> link = t ;
    }
    p = prev ;
}
```



Merging LL



Program

```
# include <iostream>
using namespace std ;
int  main( )
{
    sortedll  f, s, t ;
    f.add ( 9 ) ;
    f.add ( 12 ) ;
    f.add ( 14 ) ;
    f.add ( 17 ) ;
    f.add ( 79 ) ;
    cout << "First  LL: " ;
    f.display( ) ;
    s.add ( 12 ) ;
    s.add ( 17 ) ;
    s.add ( 24 ) ;
    s.add ( 64 ) ;
    s.add ( 87 ) ;
    cout << "\nSecond LL: " ;
    s.display( ) ;
    t.merge ( f, s ) ;
    cout << "\nMerged LL: " ;
    t.display( ) ;
}
```

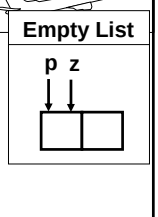
Sorted Linked List Class

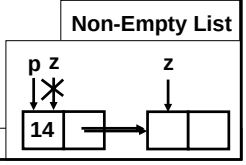
```
class sortedll
{
   private :
      struct node
      {
         int   data ;  node  *link ;
      } *p ;

   public :
      sortedll( ) ;
      ~sortedll( ) ;
      void add ( int num ) ;
      void display( ) ;
      void merge ( sortedll& l1, sortedll& l2 ) ;
} ;
```

Ctor And Dtor

```
sortedll :: sortedll( )
{
    p = NULL ;
}

sortedll :: ~sortedll( )
{
   node *q ;
   while ( p != NULL )
   {
        q = p -> link ;
        delete p ;
        p = q  ;
   }
}
```

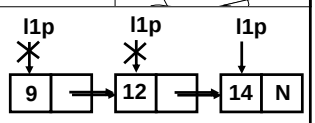
```
void sortedll :: merge ( sortedll& l1,  sortedll& l2 )
{
    node  *z ;
    node *l1p = l1.p ;
    node *l2p = l2.p ;

    if ( l1p == NULL && l2p == NULL )
         return ;

    while ( l1p != NULL && l2p != NULL )
    {
        if ( this -> p == NULL )
            z = this -> p = new node ;
        else
        {
            z -> link = new node ;
            z = z -> link ;
        }
```



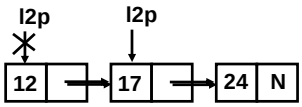


```
if ( l1p -> data < l2p -> data )
{
    z -> data = l1p -> data ;
    l1p = l1p -> link ;
}
else
{
     if ( l1p->data == l2p -> data )
     {
         z -> data = l2p -> data ;
         l1p = l1p -> link ;
         l2p = l2p -> link ;
     }
     else
     {
         if ( l2p->data < l1p->data )
         {
             z -> data = l2p -> data ;
             l2p = l2p -> link ;
         }
     } } }// while
```

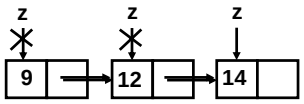
First List



Second List

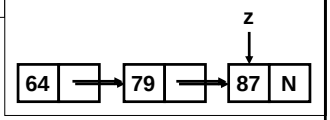


Third List




```
        while ( l1p != NULL )
        {
             z -> link = new node ;
             z = z -> link ;
             z -> data = l1p -> data
             l1p = l1p -> link ;
        }
        while ( l2p != NULL )
        {
             z -> link = new node ;
             z = z -> link ;
             z -> data = l2p -> data ;
             l2p = l2p -> link ;
        }
        z -> link = NULL ;
} // merge( )
```

Third List



Linked List & Polynomials

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- How to concatenate two linked lists
- How to represent a polynomial using a LL
- How to perform polynomial addition using LL

Concatenation Of LL

```
#include <iostream>
using namespace std ;
int main( )
{
    sortedll f, s ;
    f.add ( 1 ) ;
    f.add ( 41 ) ;
    f.add ( 3 ) ;
    f.add ( 9 ) ;
    cout << "\nFirst LL: " ;
    f.display( ) ;
    s.add ( 7 ) ;
    a.add ( 13 ) ;
    s.add ( 2 ) ;
    s.add ( 84 ) ;
    cout << "\nSecond LL: " ;
    s.display( ) ;
    f.concat ( s ) ;
    cout << "\nConcatenated LL: " ;
    f.display( ) ;
}
```

sortedll Class

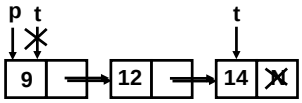
```
class sortedll
{
   private :
      struct node
      {
          int   data ;
          node  *link ;
      } *p ;

   public :
      sortedll( ) ;
      ~sortedll( ) ;
      void add ( int num ) ;
      void display( ) ;
      void concat ( sortedll& l2 ) ;
} ;
```

Ctor And Dtor

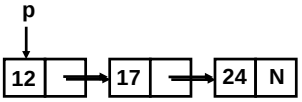
```
sortedll :: sortedll( )
{
   p = NULL ;
}

sortedll :: ~sortedll( )
{
   node *q ;
   while ( p != NULL )
   {
        q = p -> link ;
        delete p ;
        p = q ;
   }
}
```

```
void sortedll :: concat ( sortedll& l2 )
{
    node  *t , *n ;
    if ( l2.p == NULL )
        return ;
    t = p ;
    while ( t -> link != NULL )
        t = t -> link ;
    n = new node ;
    *n = *l2.p ;
    n -> link = NULL ;
    t -> link = n ;
    t = l2.p -> link ;
    while ( t != NULL )
    {
       n -> link = new node ;
       n = n -> link ;
       *n = * t ;
       n -> link = NULL ;
       t = t -> link ;
    }
}
```

First List



Second List



Polynomial Addition

```
# include <iostream>
using namespace std ;
int main( )
{
    poly p1 ;
    p1.append ( 1.4, 5 ) ;
    p1.append ( 1.5, 4 ) ;
    p1.append ( 1.7, 2 ) ;
    p1.append ( 1.8, 1 ) ;
    p1.append ( 1.9, 0 ) ;
    p1.display( ) ;
    poly p2, p3 ;
    p2.append ( 1.5, 6 ) ;
    p2.append ( 2.5, 5 ) ;
    p2.append ( -3.5, 4 ) ;
    p2.append ( 4.5, 3 ) ;
    p2.append ( 6.5, 1 ) ;
    p2.display( ) ;
    p3.add ( p1, p2 ) ;
    p3.display( ) ;
}
```

$1.4\ X^5 + 1.5\ X^4 + 1.7\ X^2 + 1.8\ X^1 + 1.9\ X^0$

$1.5\ X^6 + 2.5\ X^5 - 3.5\ X^4 + 4.5\ X^3 + 6.5\ X^1$

poly Class

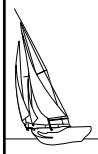
```
class poly
{
  private :
    struct polynode
    {
      float coeff ;
      int exp ;
      polynode *link ;
    } *p ;
  public :
    poly( ) ;
    ~poly( ) ;
    void append ( float c, int e ) ;
    void display( ) ;
    void addition ( poly &l1, poly &l2 ) ;
} ;
```

Ctor And Dtor

```
poly :: poly( )
{
  p = NULL ;
}

poly :: ~poly( )
{
  polynode *q ;

  while ( p != NULL )
  {
      q = p -> link ;
      delete p ;
      p = q ;
  }
}
```

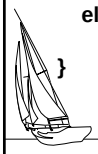
Append

```
void poly :: append ( float c, int e )
{
   polynode *t = p ;
   if ( t == NULL )
   {
      t = new polynode ;
      p = t ;
   }
   else
   {
      while ( t -> link != NULL )
         t = t -> link ;
      t -> link = new polynode ;
      t = t -> link ;
   }
   t -> coeff = c ;
   t -> exp = e ;
   t -> link = NULL ;
}
```

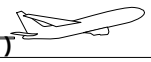


Display

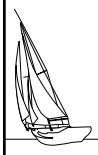
```
void poly :: display( )
{
   polynode *t = p ;
   int f = 0 ;
   while ( t != NULL )
   {
      if ( f != 0 )
      {
         if ( t -> coeff > 0 )
            cout << " + " ;
         else
            cout << " " ;
      }
      if ( t -> exp != 0 )
         cout << t -> coeff
           << "x^" << t -> exp ;
      else
         cout << t -> coeff ;
      t = t -> link ;
      f = 1 ;
   }
}
```



Addition



```
void poly :: addition ( poly &l1, poly &l2 )
{
   polynode *z ;
   polynode *l1p, *l2p ;
   l1p = l1.p ;
   l2p = l2.p ;

   if ( l1.p == NULL && l2.p == NULL )
      return ;

   while ( l1p != NULL && l2p != NULL )
   {
      if ( p == NULL )
         z = p = new polynode ;
      else
      {
         z -> link = new polynode ;
         z = z -> link ;
      }
```



```
if ( l1p -> exp < l2p -> exp )
{
   z -> coeff = l2p -> coeff ;   z -> exp = l2p -> exp ;
   l2p = l2p -> link ;
}
else
{
   if ( l1p -> exp > l2p -> exp )
   {
      z -> coeff = l1p -> coeff ;   z -> exp = l1p -> exp ;
      l1p = l1p -> link ;
   }
   else
   {
      if ( l1p -> exp == l2p -> exp )
      {
         z -> coeff = l1p -> coeff + l2p -> coeff ;
         z -> exp = l1p -> exp ;
         l1p = l1p -> link ;   l2p = l2p -> link ;
      }
   } } }
```

...Contd.

```
while ( l1p != NULL )
{
   if ( p == NULL )
   {
      p = new polynode ;
      z = p ;
   }
   else
   {
      z -> link = new polynode ;
      z = z -> link ;
   }
   z -> coeff = l1p -> coeff ;
   z -> exp = l1p -> exp ;
   l1p = l1p -> link ;
}
```

```
while ( l2p != NULL )
{
   if ( p == NULL )
   {
      p = new polynode ;
      z = p ;
   }
   else
   {
      z -> link = new  polynode ;
      z = z -> link ;
   }
   z -> coeff = l2p -> coeff ;
   z -> exp = l2p -> exp ;
   l2p = l2p -> link ;
}
z -> link = NULL ;
}
```

Circular Linked List

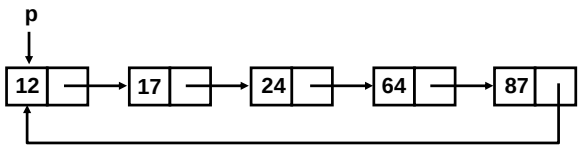
Yashavant Kanetkar

kanetkar@ksetindia.com

Objectives

- What are circular linked lists
- How to add a new node to a circular list
- How to delete an existing node from a circular list
- How to count the nodes of the circular linked list
- How to display the nodes in the circular linked list

Circular Linked List



CLL Operations

```
# include <iostream>
using namespace std ;
int main( )
{
    cll l ;
    int  c ;

    l.append ( 10 ) ;
    l.append ( 18 ) ;

    l.addatbeg ( 5 ) ;
    l.addatbeg ( 15 ) ;

    l.insert ( 2, 99 ) ;
    l.insert ( 66, 88 ) ;

    l.display( ) ;

    l.del ( 15 ) ;
    l.del ( 10 ) ;
    l.del ( 88 ) ;

    l.display( ) ;

    c = l.count( ) ;
    cout << "No. of ele. << c ;
}
```

cll Class

```
class cll
{
   private :
      struct node
      {
         int   data ;  node  *link ;
      } *p ;
   public :
      cll( ) ;
      ~cll( ) ;
      void append ( int n ) ;
      void addatbeg ( int n ) ;
      void insert ( int pos, int n ) ;
      void display( ) ;
      int count( ) ;
      void del ( int num ) ;
      void deleteall ( ) ;
} ;
```

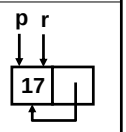
Ctor And Dtor

```
cll :: cll( )
{
   p = NULL ;
}

cll :: ~cll( )
{
   deleteall ( ) ;
}
```

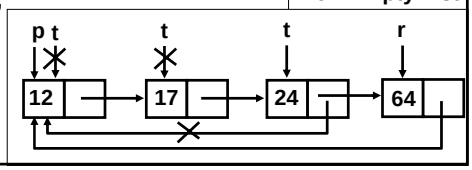
Append

```
void cll :: append ( int  n )
{
   node  *r, *t ;
   r = new node ;
   r -> data = n ;
   if ( p == NULL )
      p = r ;
   else
   {
      t = p ;
      while ( t -> link != p )
         t = t -> link ;
      t -> link = r ;
   }
   r -> link = p ;
}
```

Empty List



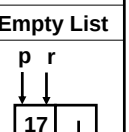
Non-Empty List



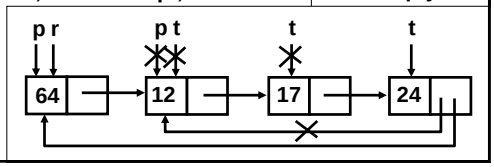
Prepend

```
void cll :: addatbeg ( int  n )
{
    node  *r , *t ;
    r = new node ;
    r -> data = n ;
    if ( p == NULL )
       r -> link = r ;
    else
    {
       t = p ;
       while ( t -> link != p )
          t = t -> link ;
       t -> link = r ;  r -> link = p ;
    }
    p = r ;
}
```

Empty List

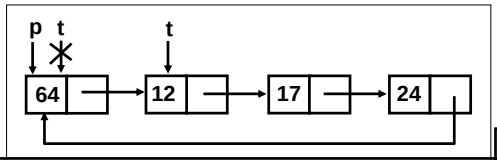


Non-Empty List



Insert

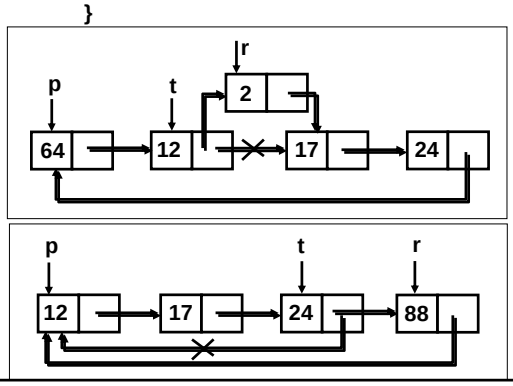
```
void cll :: insert ( int  pos, int  n )
{
    node  *t ;
    int  i ;
    t = p ;
    for ( i = 0 ; i < pos ; i++ )
    {
       if ( t -> link == p )
          break ;
       t = t -> link ;
    }
```



Cont…

Cont…

```
            r = new node ;
            r -> data = n ;
            r -> link = t -> link ;
            t -> link = r ;
        }
```

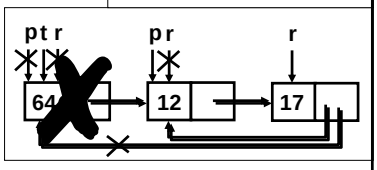


Delete Node

```
void cll :: del ( int  n )
{
    node  *t, *prev, *r ;
    t = p ;
    do
    {
        if ( t -> data == n )
        {
            if ( t == p )
            {
                r = p ;
                while ( r -> link != p )
                    r = r -> link ;
                p =  r -> link = t -> link ;
                delete t ;
            }
```

Cont…

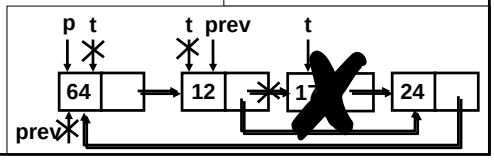
Node to be Deleted = 64



Cont…

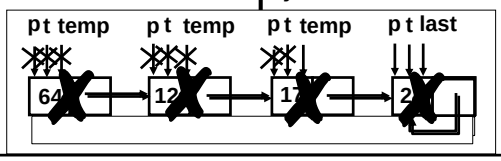
```
            else    // if  t == p
            {
                prev -> link = t -> link ;
                delete t ;
            }
            return ;
        }  // if  t -> data == n
        prev = t ;
        t = t -> link ;
    }  while ( t != p )
    cout << "Element Not Found." ;
}  // del
```

Node to be Deleted = 17

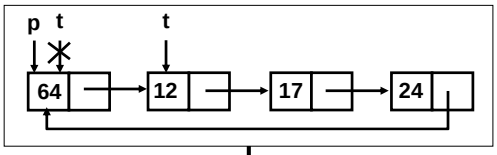


Delete All Nodes

```
void cll :: deleteall( )
{
    node  *t , *last , *temp ;
    if ( p == NULL )
        return ;
    t = p ;
    while ( t -> link != p )
        t = t -> link ;
    last = t ;
    t = p ;
    while ( t -> link != p )
    {
        temp = t ;
        t = t -> link ;
        last -> link = p = t ;
        delete temp ;
    }
    delete p ;
    p = NULL ;
}
```



Display And Count




```
void cll :: display( )
{
    node *t ;
    if ( p == NULL )
        return ;
    t = p ;
    do
    {
        cout << t -> data << "  " ;
        t = t -> link ;
    } while ( t != p ) ;
}
```

```
int cll :: count( )
{
    node *t = p ;
    int c = 0 ;
    do
    {
        t = t -> link ;
        c++ ;
    }  while ( t != p ) ;
    return c ;
}
```

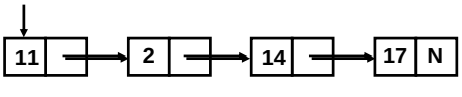
Doubly Linked List

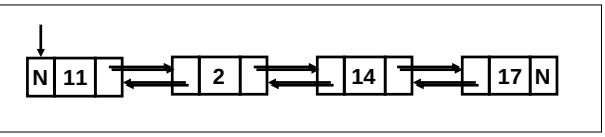
Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- Ω What are Doubly Linked Lists
- Ω How are they different than Singly Linked Lists
- Ω How to perform different operations on a Doubly Linked List

Doubly Linked List






```
struct  dnode
{
    dnode  *prev ;
    int  data ;
    dnode  *next ;
}
```

DLL Operations

```
# include <iostream>
using namespace std ;
int main( )
{
   int c ;
   dlinkedlist l ;
   l.append ( 11 ) ;
   l.append (  2 ) ;
   l.append ( 14 ) ;
   l.append ( 17 ) ;
   l.append ( 99 ) ;
   l.display( ) ;
   l.addatbeg ( 33 ) ;
   l.addatbeg ( 55 ) ;
   l.display( ) ;
   l.insert ( 4, 66 ) ;
   l.insert ( 2, 96 ) ;
   l.display( ) ;
   c = l.count( ) ;
   cout <<  "count = " << c ;

   l.del ( 55 ) ;
   l.del ( 2 ) ;
   l.del ( 99 ) ;
   l.display( ) ;
   c = l.count( ) ;
   cout <<  "count = " << c ;
   return 0 ;
}
```

dlinkedlist Class

```
class dlinkedlist
{
   private :
      struct dnode
      {
         dnode *prev ;  int data ;  dnode * next ;
      } *p ;
   public :
      dlinkedlist( ) ;
      ~dlinkedlist( ) ;
      void append ( int n ) ;
      void addatbeg ( int n ) ;
      void insert ( int pos, int n ) ;
      void display( ) ;
      int count( ) ;
      void del ( int i ) ;
} ;
```

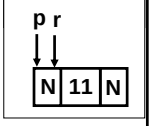
Ctor And Dtor

```
dlinkedlist :: dlinkedlist( )
{
   p = NULL ;
}


dlinkedlist :: ~dlinkedlist( )
{
   dnode *t = p ;
   while ( p != NULL )
   {
      t = p -> next ;
      delete p ;
      p = t ;
   }
}
```

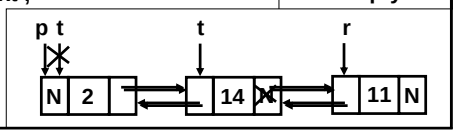
Append

```
void dlinkedlist :: append ( int  n )
{
   dnode  *r, *t ;
   r = new  dnode ;
   r -> data = n ;   r -> next = NULL ;
   if ( p == NULL )
   {
       r -> prev = NULL ;   p = r ;
   }
   else
   {
       t = p ;
       while ( t -> next != NULL )
           t = t -> next ;
       r -> prev = t ;
       t -> next = r ;
   }
}
```

Empty List

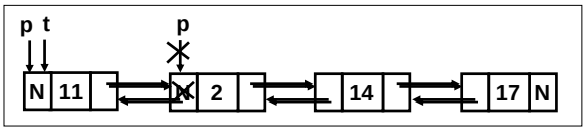


Non-Empty List



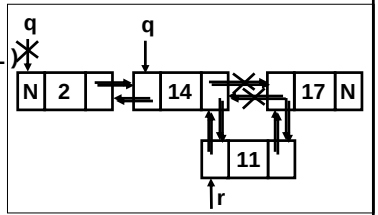
Prepend

```
void dlinkedlist :: addatbeg ( int  n )
{
     dnode  *t ;
     t = new  dnode ;
     t -> prev = NULL ;
     t -> data = n ;
     t -> next = p ;
     p -> prev = t ;
     p = t ;
}
```



Insert

```
void dlinkedlist :: insert ( int  pos, int  n )
{
   int  i ;
   dnode  *r, *q ;
   q = p ;
   for ( i = 0 ; i < pos ; i++ )
   {
       if ( q -> next == NULL )
           break ;
       q = q -> next ;
   }
   r = new  dnode ;
   r -> data = n ;
   r -> prev = q ;   r -> next = q -> next ;
   if ( q -> next != NULL )
       q -> next -> prev = r ;
   q -> next = r ;
}
```

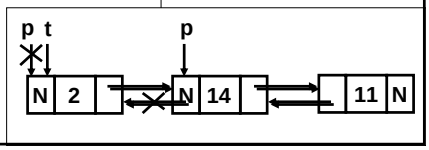


What if q points to last node

Delete

```
void dlinkedlist :: del ( int  n )
{
    dnode  *t ;
    t = p ;
    while ( t != NULL )
    {
        if ( t -> data == n )
        {
            if ( t == p )
            {
                p = p -> next ;
                p -> prev = NULL ;
            }
```

Node to be Deleted = 2

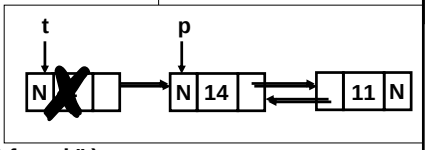


Cont…

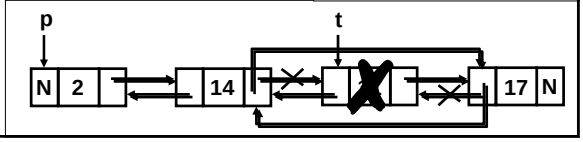
Cont…

```
            else
            {
                t -> prev -> next = t -> next ;
                if ( t -> next != NULL )
                    t -> next -> prev = t -> prev ;
            }
            delete t ;
            return ;
        } // if
        t = t -> next ;
    } // while
    printf ( "\nNo. not found." ) ;
} // d_delete
```

Node to be Deleted = 2



Node to be Deleted = 11



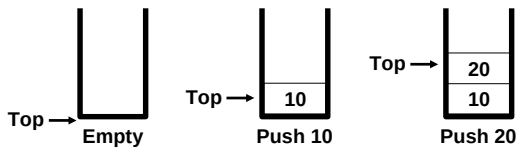
Stack

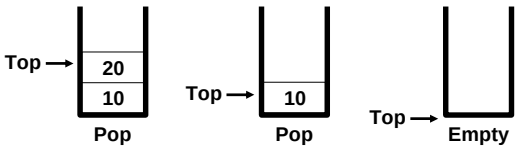
Asang Dani
asang@ksetindia.com

Objectives

- Stack and its utilities
- Stack as an Array
- Push and Pop operations
- Displaying stack elements

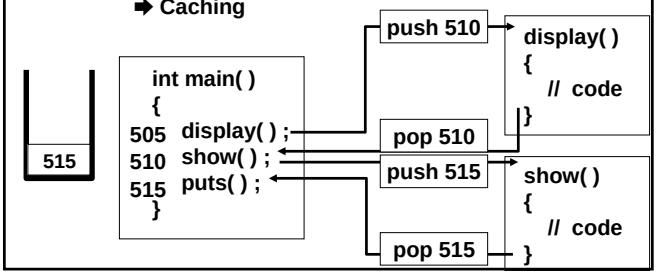
Stack





Utility

- Store local variables in a function
- Manage function calls
- Conversion of Expressions
- Evaluation of Expressions
- Caching

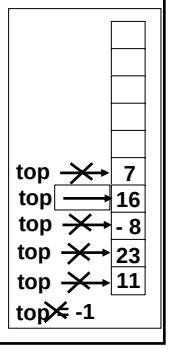


Program

```
#include <iostream>

using namespace std ;

const int MAX = 10 ;

class stack
{
 private :
   int arr [ MAX ] ;
   int top ;

public :
   stack ( ) ;
   void push ( int item ) ;
   int pop( ) ;
   void display ( ) ;
   int count ( ) ;
} ;
```

Operations of Stack

```
int main ( )
{
  stack s ;
  s.push ( 11 ) ;
  s.push ( 23 ) ;
  s.push ( -8 ) ;
  s.push ( 16 ) ;
  s.push ( 7 ) ;
  cout << "\n\nItem popped: " << s.pop ( ) ;

  cout << "\nNo. of items : " << s.count ( ) ;
  cout << endl ;

  s.display ( ) ;
}
```



Implementation

```
stack :: stack( )
{
  top = -1 ;
}

void stack :: push ( int item )
{
  if ( top == MAX - 1 )
  {
    cout << "Stack is full" ;
    return ;
  }
  top ++ ;
  arr[ top ] = item ;
}
```

```
int stack :: pop( )
{
  if ( top == -1 )
  {
    cout << "Stack is empty" ;
    return -1 ;
  }
  int data = arr[ top ] ;
  top -- ;
  return data ;
}
```

Implementation (2)

```
void stack :: display ( )
{
  for ( int i = 0 ; i <= top ; i ++ )
    cout << arr [ i ] << "\t" ;
  cout << endl ;
}

int stack :: count ( )
{
  return top + 1 ;
}
```

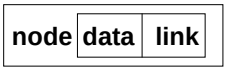
Stack as a Linked List

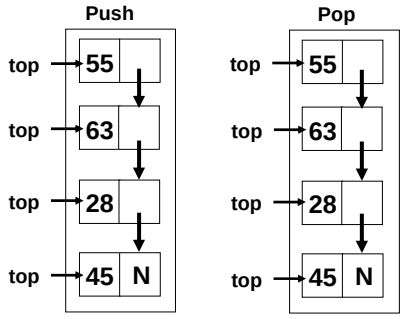
Asang Dani
asang@ksetindia.com

Objectives

- Why Arrays are not enough
- Advantages of linked lists
- Stack as a Linked List
- Operations on Stack

Stack As A Linked List





Program

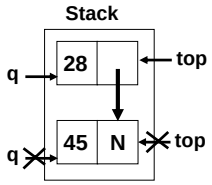
```
#include <iostream>

using namespace std ;

int main( )
{
   stack s ;
   int  t, item ;

   s.push ( 45 ) ;
   s.push ( 28 ) ;
   s.push ( 63 ) ;
   s.push ( 55 ) ;

   s.display ( ) ;
   t = s.count ( ) ;
   cout << "Total items: " << t ;

   item = s.pop ( ) ;
   cout << "\n\nItem: " << item ;

   s.display ( ) ;
   t = s.count ( ) ;
   cout << "Total items: " << t ;
}
```

stack class

```
class stack
{
 private :
   struct node
   {
     int data ;
     node *link ;
   } * top ;
 public :
   stack( ) ;
   void push ( int item ) ;
   int pop( ) ;
   void display ( ) ;
   int count ( ) ;
   ~stack( ) ;
} ;
stack :: stack( )
{
 top = NULL ;
}
```

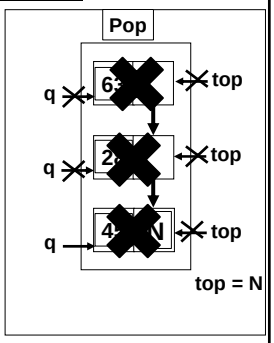
Push Operation

```
void stack :: push ( int  item )
{
   node  *q ;
   q = new node ;
   q -> data = item ;
   q -> link = top ;
   top = q ;
}
```



Pop Operation

```
int stack :: pop ( )
{
    node  *q ;  int  item ;
    if ( top == NULL )
    {
       cout << "Stack is empty" ;
       return  -1 ;
    }
    q = top ;
    item = q -> data ;
    top = q -> link ;
    delete q ;
    return ( item ) ;
}
```



Display And Count

```
void  stack :: display ( )
{
   node * q = top ;
   while ( q != NULL )
   {
      cout << q -> data << "\t" ;
      q = q -> link ;
   }
}
```

```
int  stack :: count ( )
{
   node * q = top ; int  c = 0 ;
   while ( q != NULL )
   {
      q = q -> link ;
      c++ ;
   }
   return  c ;
}
```

```
stack :: ~stack( )
{
  node *q ;
  while ( top != NULL )
  {
    q = top ;
    top = top -> link ;
    delete q ;
  }
}
```

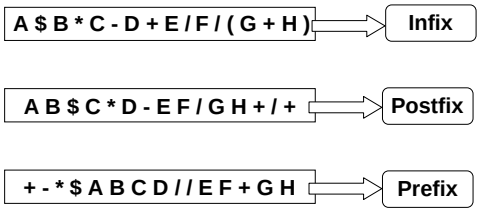
Stack Expressions

Yashavant Kanetkar

Objectives

- **Expressions of different forms**

Expressions



In To Post

Scan from L to R. Repeat step 1- 4

Token	Operation
operand	Add to expression
(	Push to stack
operator	Pop oper. If P(Popped) >= P(Scanned) add to expr. Push scanned operator to stack
)	Pop till (. Add popped token to expr. Delete)

Pop stack elements if any and add to expression

A + (B * C - (D / E $ F) * G) * H

Tok.	Stack	Expression
A	Empty	A
+	+	A
(	+ (	A
B	+ (	A B
*	+ (*	A B
C	+ (*	A B C
-	+ (-	A B C *
(	+ (- (	A B C *
D	+ (- (	A B C * D
/	+ (- (/	A B C * D
E	+ (- (/	A B C * D E
$	+ (- (/ $	A B C * D E

...Contd.

A + (B * C - (D / E $ F) * G) * H

Tok.	Stack	Expression
$	+ (- (/ $	A B C * D E
F	+ (- (/ $	A B C * D E F
)	+ (-	A B C * D E F $ /
*	+ (- *	A B C * D E F $ /
G	+ (- *	A B C * D E F $ / G
)	+	A B C * D E F $ / G * -
*	+ *	A B C * D E F $ / G * -
H	+ *	A B C * D E F $ / G * - H
		A B C * D E F $ / G * - H * +

In To Post

A $ B * C - D + E / F / (G + H)

Tok.	Stack	Expression
A	Empty	A
$	$	A
B	$	A B
*	*	A B $
C	*	A B $ C
-	-	A B $ C *
D	-	A B $ C * D
+	+	A B $ C * D -
E	+	A B $ C * D - E
/	+ /	A B $ C * D - E
F	+ /	A B $ C * D - E F
/	+ /	A B $ C * D - E F /
(	+ / (	A B $ C * D - E F /
G	+ / (	A B $ C * D - E F / G
+	+ / (+	A B $ C * D - E F / G
H	+ / (+	A B $ C * D - E F / G H
)	+ /	A B $ C * D - E F / G H +
		A B $ C * D - E F / G H + / +

Infix to Postfix using stack

Asang Dani
asang@ksetindia.com

Objectives

- Infix to Postfix Conversion
- Program for Infix to Postfix conversion
- Helper Functions
- Push( ) and Pop( ) functions

Program

```
#include <iostream>
#include <cctype>
using namespace std ;

int main ( )
{
  char expr [ MAX ] ;
  infix q ;
  cout << "\nEnter an expression in infix form: " ;
  cin.getline ( expr, MAX ) ;

  q.setexpr ( expr ) ;
  q.convert ( ) ;

  cout << "\nThe postfix expression is: " ;
  q.show( ) ;
}
```

Infix

A $ B * C - D + E / F / (G + H)

Postfix

A B $ C * D - E F / G H + / +

class infix

```
class infix
{
  private :
    char target [ MAX ] , stack [ MAX ] ;
    char *s, *t ;
    int top ;

  public :

    infix ( ) ;
    void setexpr ( char *str ) ;
    void push ( char c ) ;
    char pop ( ) ;
    void convert ( ) ;
    int priority ( char c ) ;
    bool isoperator ( char  ch ) ;
    void show( ) ;
} ;
```

constructor etc..

```
infix :: infix( )
{
  top = -1 ;
  strcpy ( target , "" ) ;
  strcpy ( stack , "" ) ;
  t = target ;
  s = ""  ;
}

void infix :: setexpr ( char *str )
{
  s = str ;
}
```

```
void infix :: convert ( )
{
    while ( *s != '\0' )
    {
        if ( *s == ' ' || *s == '\t' )  {
            s + + ;
            continue ;
        }
        if ( isdigit ( *s ) || isalpha ( *s ) )
        {
            while ( isdigit ( *s ) || isalpha ( *s ) )
            {
                *t = *s ;
                s + + ;  t + + ;
            }
        }
        if ( *s == '(' ) {
            push ( *s ) ;
            s + + ;
        }
```

Infix

A $ B * C - D + E / F / (G + H)

Postfix

A B $ C * D - E F / G H + / +

Contd...

...Contd.

```
char opr ;
if ( isoperator ( *s ) )
{
    if ( top != -1 )
    {
        opr = pop ( ) ;
        while ( priority ( opr ) >= priority ( *s ) )
        {
            *t = opr ;  t + + ;
            opr = pop ( ) ;
            if ( top == -1 )
              break ;
        }
        if ( opr != -1 )
          push ( opr ) ;
    }
    push ( *s ) ;
    s + + ;
}
```

Contd...

Infix: A $ B * C - D + E / F / (G + H)

F	+ /	A B $ C * D - E F
/	+ /	A B $ C * D - E F /

...Contd.

```
if ( *s == ')' )
{
    opr = pop ( ) ;
    while ( opr  !=  '(' )
    {
        *t = opr ;  t + + ;
        opr =  pop ( ) ;
    }
    s + + ;
}
} // while
while ( top != -1 )
{
    opr = pop ( ) ;
    *t = opr ;  pt + + ;
}
*t = '\0' ;
} // convert
```

H	+ / (+	A B $ C * D - E F / G H
)	+ /	A B $ C * D - E F / G H +

)	+ /	A B $ C * D - E F / G H +
		A B $ C * D - E F / G H + / +

Helper Functions

```
int infix :: priority ( char  c )
{
    if ( c == '$' )
        return  3 ;
    if ( c == '*' || c == '/' || c == '%' )
        return  2 ;
     if ( c == '+' || c == '-' )
        return  1 ;
    return  0 ;
}
```

```
int infix ::
isoperator ( char  ch )
{
    char  str[ ] = "*+/%-$" ;
    char  *p ;
    p = str ;
    while ( *p  !=  '\0' )
    {
        if ( *p == ch )
            return  1 ;
        p++ ;
    }
    return  0 ;
}
```

push( ) and pop( )

```
void infix :: push ( char  c )
{
     if ( top == MAX - 1 )
     {
          cout << "Stk. full." ;
          return ;
     }
     top++ ;
     stack [ top ] = c ;
}
```

```
char infix :: pop ( )
{
     char  item ;
     if ( top == -1 )
     {
          cout << "Stk. empty" ;
          return  -1 ;
     }
     item = stack [ top ] ;
     top-- ;
     return  item ;
}
```

Postfix Evaluation

Asang Dani
asang@ksetindia.com

Objectives

- **Evaluation of Postfix form**
- **Example of evaluation of Postfix form**
- **Program for evaluation of Postfix form**

Evaluate Postfix

Scan from L to R. Repeat step 1- 2

Token	Operation
operand	Add to stack
operator	Pop stack into n1 Pop stack into n2 Perform n3 = n2 operator n1 Push n3

Pop stack to obtain result

An Example

4 2 $ 3 * 3 - 8 4 / 1 1 + / +

Token	Stack
4	4
2	4, 2
$	16
3	16, 3
*	48
3	48, 3
-	45
8	45, 8
4	45, 8, 4
/	45, 2
1	45, 2, 1
1	45, 2, 1, 1
+	45, 2, 2
/	45, 1
+	46

Program

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <cctype>
using namespace std ;

int main( )
{
  char expr [ MAX ] ;
  cout << "\nEnter postfix expr. to be evaluated : " ;
  cin.getline ( expr, MAX ) ;

  postfix q ;

  q.setexpr ( expr ) ;
  q.calculate ( ) ;
  q.show ( ) ;
}
```

postfix class

```
const int MAX = 50 ;

class postfix
{
  private :
    int stack [ MAX ] ;
    int top, nn ;
    char *s ;

  public :

    postfix ( ) ;
    void setexpr ( char *str ) ;
    void push ( int item ) ;
    int pop( ) ;
    void calculate( ) ;
    void show( ) ;
} ;
postfix :: postfix( )
{
  top = -1 ;
}
void
postfix :: setexpr ( char *str )
{
  s = str ;
}
```

Evaluate Postfix

```
void postfix :: calculate( )
{
 int n1, n2, n3 ;
 while ( *s )
 {
  if ( *s == ' ' || *s == '\t' ) {
    s++ ;
    continue ;
  }
  if ( isdigit ( *s ) )
  {
    nn = *s - '0' ;
    push ( nn ) ;
  }
```

```
 else
 {
  n1 = pop( ) ;
  n2 = pop( ) ;
  switch ( *s )
  {
   case '+' :
      n3 = n2 + n1 ;
      break ;
   case '-' :
      n3 = n2 - n1 ;
      break ;
```

Evaluate Postfix

```
    case '/' :
       n3 = n2 / n1 ;
       break ;
    case '*' :
       n3 = n2 * n1 ;
       break ;
    case '%' :
       n3 = n2 % n1 ;
       break ;
    case '$' :
       n3 = ( int )pow ( ( double ) n2 , ( double ) n1 ) ;
       break ;
    default :
       cout << "Unknown operator" ;
       exit ( 1 ) ;
   } // switch
```

```
       push ( n3 ) ;
     } // else
     s++ ;
   } // while
  }
```

push( ) and pop( )

```
void postfix :: push ( char  c )
{
    if ( top == MAX - 1 )
    {
        cout << "Stk. full." ;
        return ;
    }
    top++ ;
    stack [ top ] = c ;
}
void postfix :: show( )
{
 nn = pop ( ) ;
 cout << "Result is: " << nn ;
}
```

```
char postfix:: pop ( )
{
    char  item ;
    if ( top == -1 )
    {
        cout << "Stk. empty" ;
        return  -1 ;
    }
    item = stack [ top ] ;
    top-- ;
    return  item ;
}
```

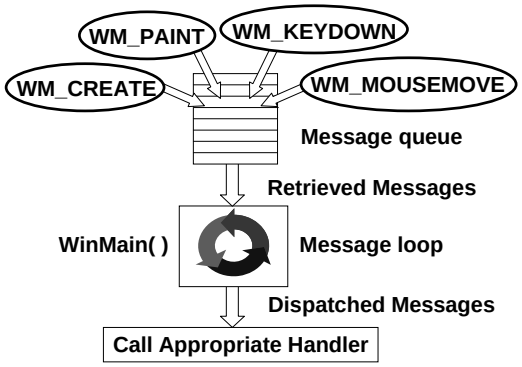
Queue

Yashavant Kanetkar
kanetkar@ksetindia.com

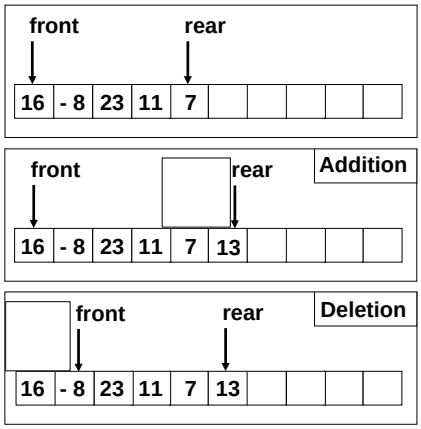
Objectives

- What are queues
- Where are they used
- Basic operations on Queues

Windows & Queue



Operations On Queue



Queue's Use

```
# include <iostream>
using namespace std ;
const int MAX = 10 ;
int main( )
{
   queue a ;
   a.addq ( 23 ) ;
   a.addq ( 9 ) ;
   a.addq ( 11 ) ;
   a.addq ( -10 ) ;
   int i = a.delq( ) ;
   cout << "\ndeleted: " << i ;
}
```

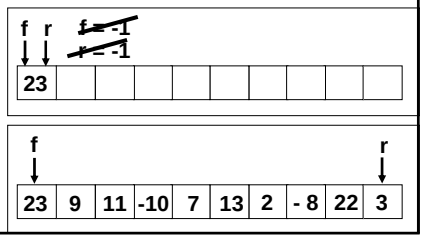
```
class queue
{
   private :
      int arr [ MAX ] ;
      int f, r ;
   public :
      queue( ) ;
      void addq ( int item ) ;
      int delq( ) ;
} ;
```

Ctor

```
// initialises data members
queue :: queue( )
{
   f = -1 ;
   r = -1 ;
}
```

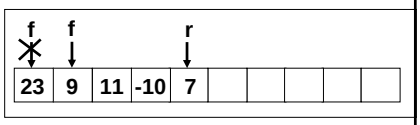
Add

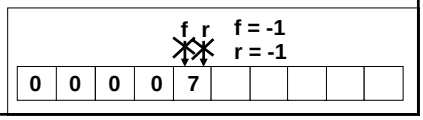
```
void queue :: addq ( int item )
{
   if ( r == MAX - 1 )
   {
      cout << "\nQueue is full" ;
      return ;
   }
   r++ ;
   arr [ r ] = item ;

   if ( f == -1 )
      f = 0 ;
}
```



Delete

```
int queue :: delq( )
{
   int data ;

   if ( f == -1 )
   {
      cout << "\nQueue is Empty" ;
      return -1 ;
   }
   data = arr [ f ] ;
   arr [ f ] = 0 ;
   if ( f == r )
      f = r = -1 ;
   else
      f++ ;
   return data ;
}
```





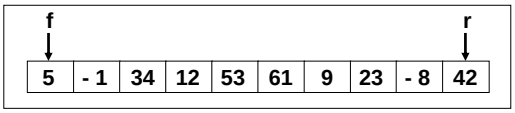
Circular Queue

Yashavant Kanetkar
kanetkar@ksetindia.com

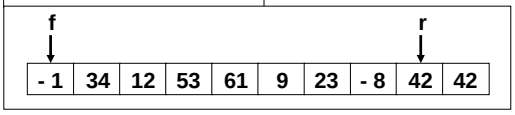
Objectives

- **Limitations of queues**
- **What are Circular Queues**
- **Implementation of Circular Queue**

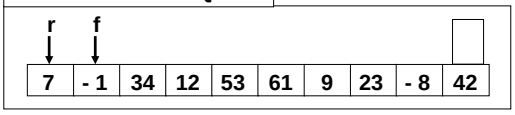
Problems



Soln. I - Shift Elements



Soln. II - Circular Queue



Using Circular Queue

```
# include <iostream>
using namespace std ;
const int MAX = 10 ;
int main( )
{
   cqueue q ;
   q.addq ( 5 ) ;
   q.addq ( -1 ) ;
   q.addq ( 34 ) ;
   q.addq ( 12 ) ;
   q.addq ( 53 ) ;
   cout << "\nElements in Q: " ;
   q.display( ) ;
   int i = q.delq( ) ;
   cout << "Item deleted: " << i ;
   cout << "\nAfter deletion: " ;
   q.display( ) ;
}
```

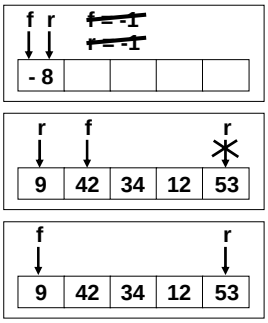
```
class cqueue
{
   private :
      int arr[ MAX ] ;
      int f, r ;
   public :
      cqueue( ) ;
      void addq ( int item )
      int delq( ) ;
      void display( ) ;
} ;
```

Constructor

```
cqueue :: cqueue( )
{
   f = r = -1 ;
   for ( int i = 0 ; i < MAX ; i++ )
      arr[ i ] = 0 ;
}
```

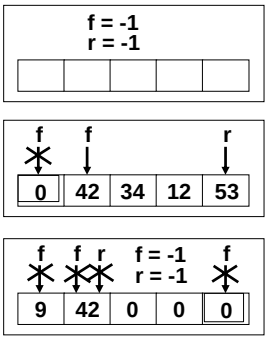
Add Element

```
void cqueue :: addq ( int  item )
{
   if ( ( r == MAX - 1 && f == 0 )  || ( r + 1 == f ) )
   {
      cout << "Queue is full." ;
      return ;
   }
   if ( r == MAX - 1 )
      r = 0 ;
   else
      r ++ ;
   arr[ r ] = item ;
   if ( f == -1 )
      f = 0 ;
}
```



Delete Element

```
int cqueue :: delq( )
{
   int  i ;
   if ( f == -1 )
   {
      cout << "Queue is empty" ;
      return  -1 ;
   }
   i = arr[ f ] ;   arr[ f ] = 0 ;
   if ( f == r )
      f = r = -1 ;
   else
   {
      if ( f == MAX - 1 )
         f = 0 ;
      else
         f ++ ;
   }
   return  i ;
}
```



Display

```
void queue :: display( )
{
   cout << endl ;
   for ( int i = f ; i <= r ; i++ )
      cout << arr[ i ] << "  " ;
   cout << endl ;
}
```

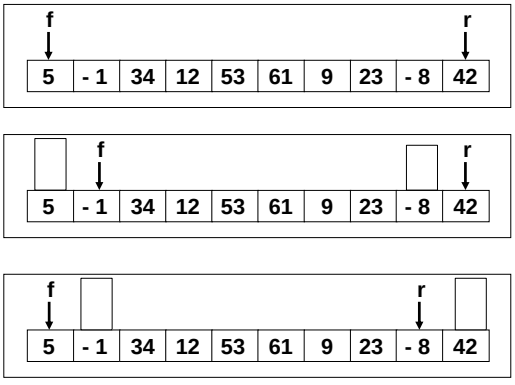
Deque & Priority Queue

Yashavant Kanetkar
kanetkar@ksetindia.com

Objectives

- **What is a Dequeue**
- **Operations on a Dequeue**
- **What is a priority queue**

Deque



Program

```
# include <iostream>
using namespace std ;
const int MAX = 10 ;
int main( )
{
   dque a ;
   a.addqatend ( 17 ) ;
   a.addqatbeg ( 10 ) ;
   a.addqatend ( 8 ) ;
   a.addqatbeg ( -9 ) ;
   a.addqatend ( 13 ) ;
   a.display( ) ;
   int n = a.count( ) ;
   cout << "\nCount = " << n ;
   cout << "\nItem extracted: " ;
   cout << a.delqatbeg( ) ;
   cout << "\nAfter deletion: " ;
   a.display( ) ;

   cout << "\nItem extracted: " ;
   cout << a.delqatend( ) ;
   cout << "\nAfter deletion: " ;
   a.display( ) ;
}
```

dque Class

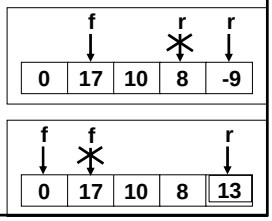
```
class dque
{
   private :
     int arr[ MAX ] ;
     int f, r ;

   public :
     dque( ) ;
     void addqatbeg ( int item ) ;
     void addqatend ( int item ) ;
     int delqatbeg( ) ;
     int delqatend( ) ;
     void display( ) ;
     int count( ) ;
} ;
```

Ctor

```
dque :: dque( )
{
   f = r = -1 ;
   for ( int i = 0 ; i < MAX ; i++ )
      arr[ i ] = 0 ;
}
```

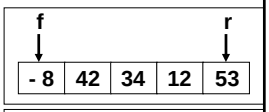
Add At End

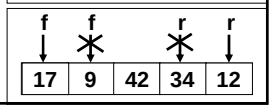
```
void dque :: addqatend ( int  item )
{
   if ( f == 0 && r == MAX - 1 )
   {
      cout << "Deque is full." ;
      return ;
   }
   if ( f == -1 )
   {
      r = f = 0 ;
      arr[ r ] = item ;
      return ;
   }
   if ( r == MAX - 1 )
   {
      for ( int i = f - 1 ; i < r ; i++ )
         arr[ i ] = arr[ i + 1 ] ;
         r- - ;  f- - ;
      }
      r++ ;
      arr[ r ] = item ;
}
```



Add At Begin

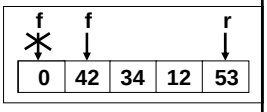
```
void dque :: addqatbeg ( int  item )
{
   if ( f == 0 && r == MAX-1 )
   {
      cout << "Deque is full." ;
      return ;
   }
   if ( f == -1 )
   {
      f = r = 0 ;
      arr[ f ] = item ;
      return ;
   }
   if ( r != MAX - 1 )
   {
      int c = count( ) ;
      int k = r + 1 ;
      for ( int i = 1 ; i <= c ; i++ )
      {
         arr[ k ] = arr[ k - 1 ] ;
         k- - ;
      }
      arr[ k ] = item ;
      f = k ; r++ ;
   }
   else
   {
      f- - ;
      arr[ f ] = item ;
   }
}
```

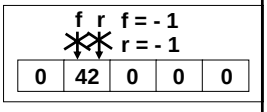




Delete From Front

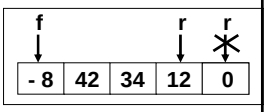
```
int dque :: delqatbeg( )
{
   int  item ;
   if ( f == -1 )
   {
      cout << "Deque is empty." ;
      return  0 ;
   }
   item = arr[ f ] ;
   arr[ f ] = 0 ;
   if ( f == r )
      f = r = -1 ;
   else
      f++ ;
   return  item ;
}
```

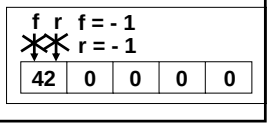




Delete From End

```
int dque :: delqatend( )
{
   int  item ;
   if ( f == -1 )
   {
      cout << "Deque is empty" ;
      return  0 ;
   }
   item = arr[ r ] ;
   arr[ r ] = 0 ;
   r-- ;
   if ( r == -1 )
      f = -1 ;
   return  item ;
}
```





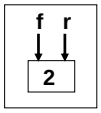
Count And Display

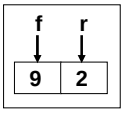
```
void dque :: display( )
{
   cout << endl << "f->";
   for ( int i = f ; i <= r ; i++ )
      cout << "  " << arr[ i ] ;
   cout << " <-r" ;
}
```

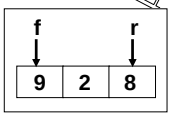
```
int dque :: count( )
{
   int c = 0 ;
   for ( int i = f ; i <= r ; i++ )
   {
      if ( arr[ i ] != 0 )
         c++ ;
   }
   return c ;
}
```

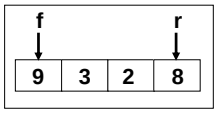
Priority Queue

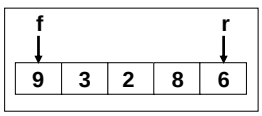
Job	Priority
2	2
9	4
8	2
3	3
6	1
1	4
7	5
4	4
0	4
5	2











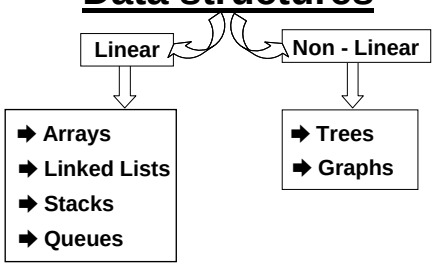
Trees

Asang Dani

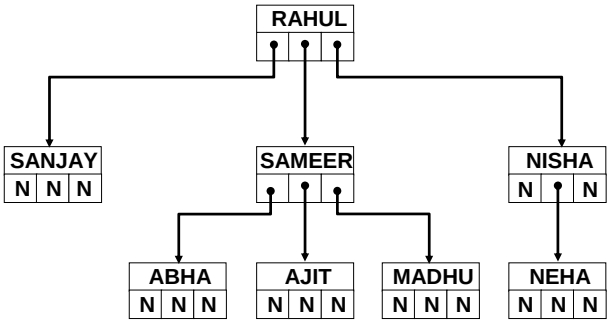
Objectives

- Trees
- Trees terminology
- Types of Trees

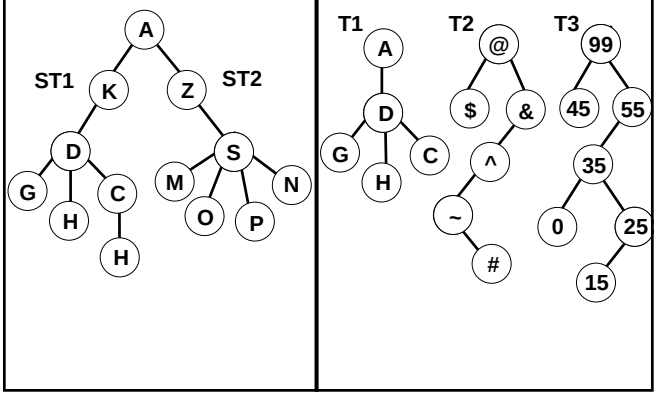
Data structures



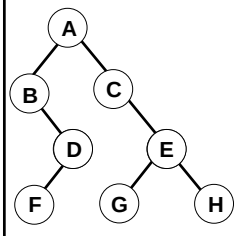
Trees



General Trees And Forest

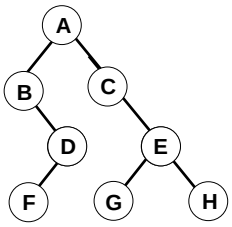


Terminology



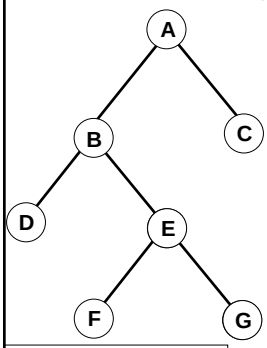
- Binary Tree - Each node has 0, 1, 2 children
- A - Root node
- B, D, F - Nodes of left sub-tree
- C, E, G, H - Nodes of right sub-tree
- F, G, H - Leaf nodes
- A is Father of B & C
- E is right child C
- G is right descendant of A
- F is left child of D
- F is left descendant of A
- A is ancestor of D
- G & H are siblings, D and E aren't

More Terminology



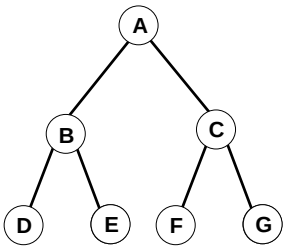
- Degree of a node is no. of nodes connected to it
- Level of root node is 0
- Level of any other node is 1 more than level of its father
- Depth of a binary tree is maximum level of a node

Strictly And Complete



Strictly Binary Tree

Each non-leaf node has 2 children



Complete Binary Tree

Strictly binary tree with all leaf nodes at same level

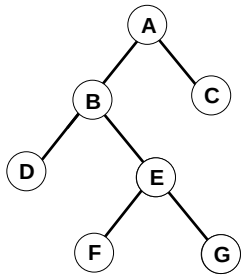
Tree Traversal

Asang Dani

Objectives

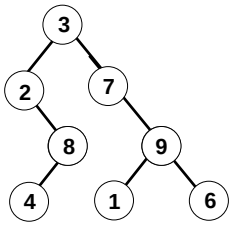
- Traversal of Trees
- Reconstruction of Trees

Traversal



Pre-order Traversal - Root - Left - Right	A, B, D, E, F, G, C
In-order Traversal - Left - Root - Right	D, B, F, E, G, A, C
Post-order Traversal - Left - Right - Root	D, F, G, E, B, C, A

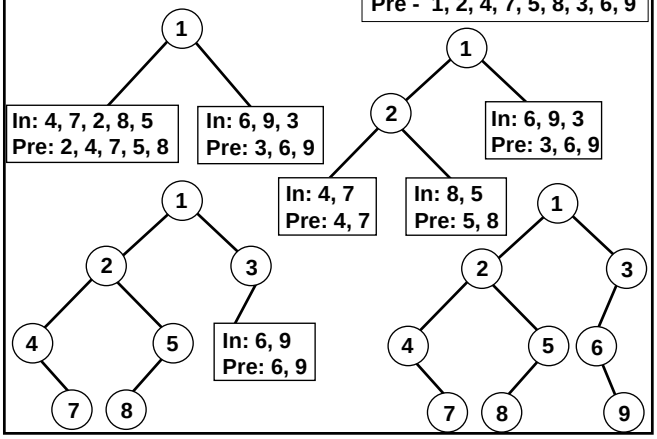
Exercise



Pre-order Traversal - Root - Left - Right	3, 2, 8, 4, 7, 9, 1, 6
In-order Traversal - Left - Root - Right	2, 4, 8, 3, 7, 1, 9, 6
Post-order Traversal - Left - Right - Root	4, 8, 2, 1, 6, 9, 7, 3

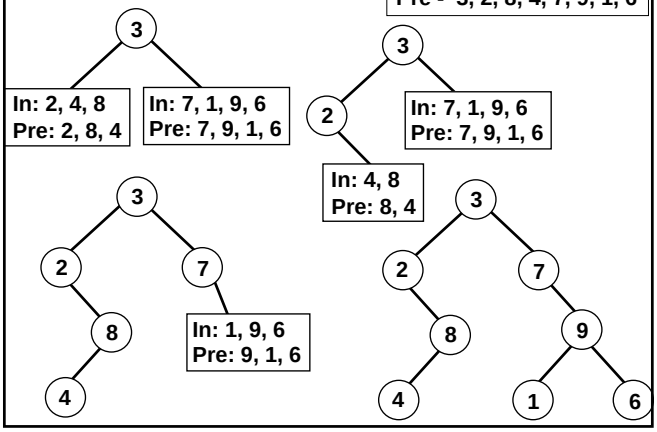
Reconstruction - I

In - 4, 7, 2, 8, 5, 1, 6, 9, 3
Pre - 1, 2, 4, 7, 5, 8, 3, 6, 9



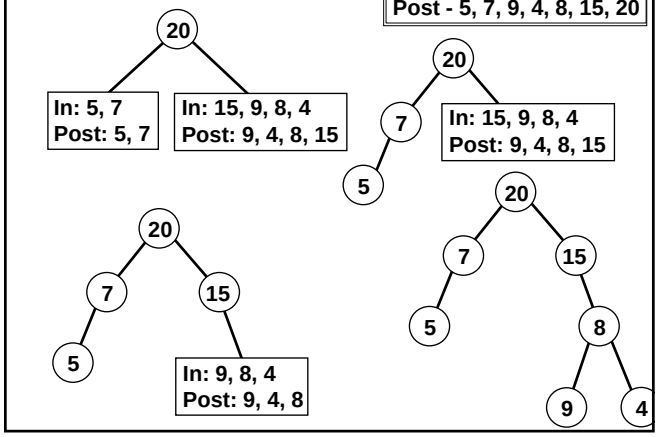
Exercise

In - 2, 4, 8, 3, 7, 1, 9, 6
Pre - 3, 2, 8, 4, 7, 9, 1, 6



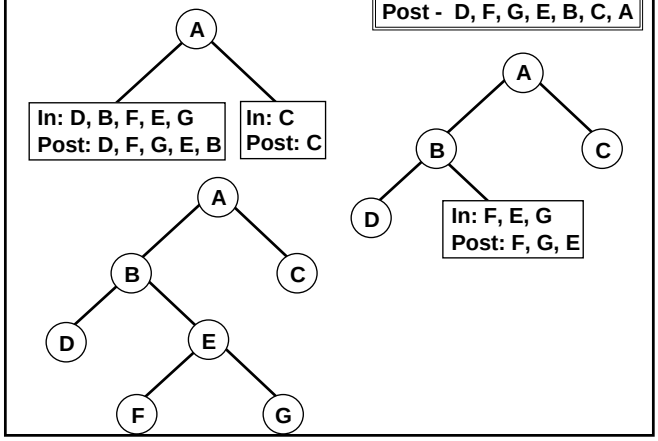
Reconstruction - II

In - 5, 7, 20, 15, 9, 8, 4
Post - 5, 7, 9, 4, 8, 15, 20



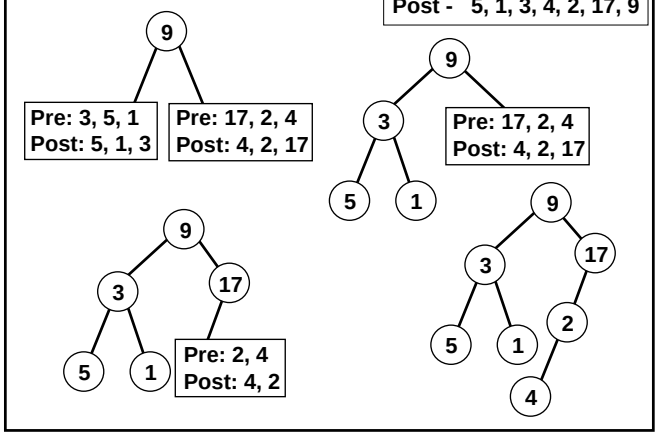
Exercise

In - D, B, F, E, G, A, C
Post - D, F, G, E, B, C, A



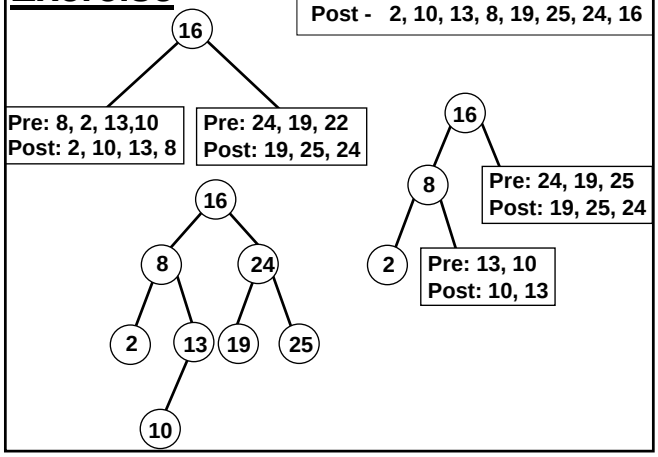
Reconstruction - III

Pre - 9, 3, 5, 1, 17, 2, 4
Post - 5, 1, 3, 4, 2, 17, 9



Exercise

Pre - 16, 8, 2, 13, 10, 24, 19, 25
Post - 2, 10, 13, 8, 19, 25, 24, 16



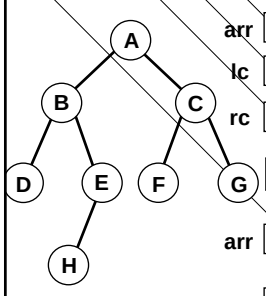
BST - I

Asang Dani

Objectives

- Representing trees using arrays & linked lists
- What are binary search trees

Array Representation



lc = 2 pos + 1
rc = 2 pos + 2

3 - Arrays

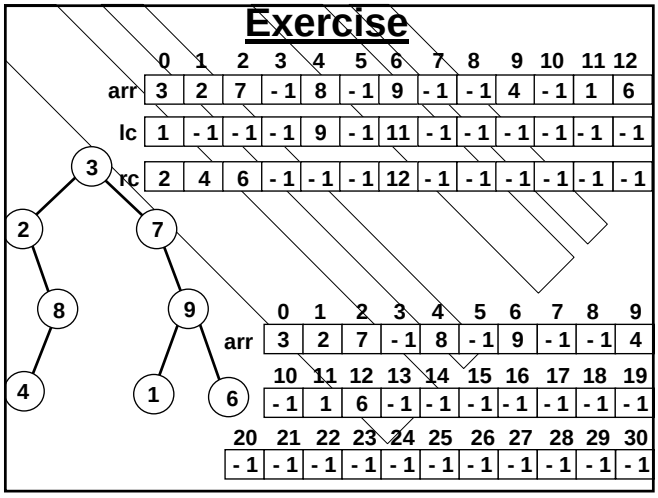
	0	1	2	3	4	5	6	7	8	9
arr	A	B	C	D	E	F	G	'\0'	'\0'	H
lc	1	3	5	- 1	9	- 1	- 1	- 1	- 1	- 1
rc	2	4	6	- 1	- 1	- 1	- 1	- 1	- 1	- 1

1 - Array

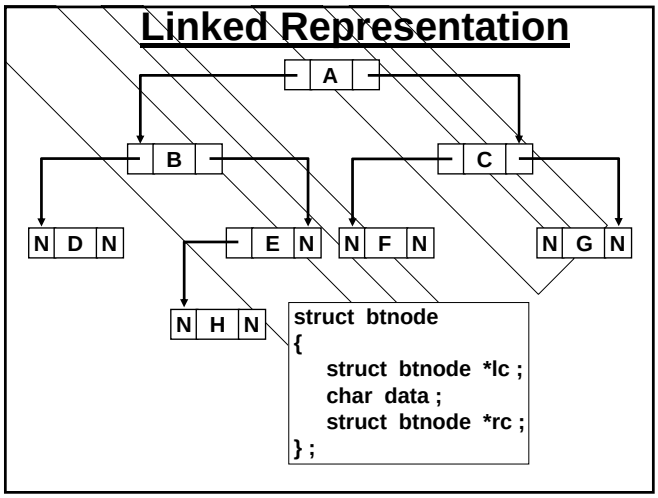
$2^{(\text{depth} + 2)} - 1$

	0	1	2	3	4	5	6	7	8	9
arr	A	B	C	D	E	F	G	'\0'	'\0'	H
	10	11	12	13	14	15	16	17	18	19
	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
	20	21	22	23	24	25	26	27	28	29	30
	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

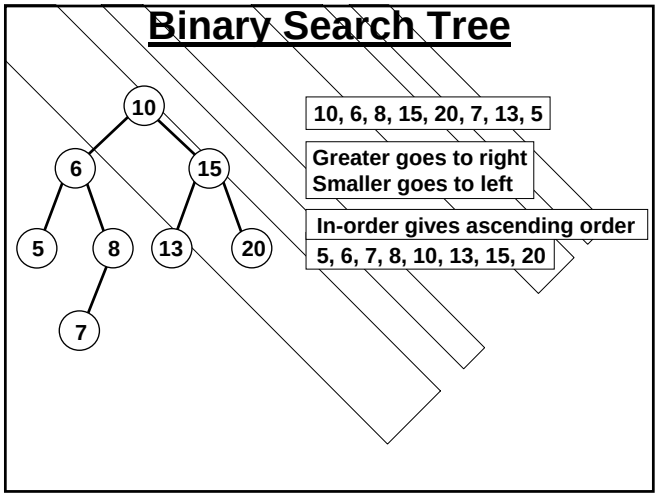
Exercise



Linked Representation

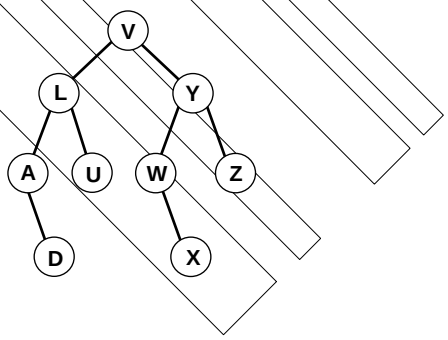


Binary Search Tree



Exercise

V, L, A, Y, U, W, D, Z, X



Binary Search Trees - II

Asang Dani
asang@ksetindia.com

Objectives

- Inserting elements in a BST
- Inorder, preorder & postorder traversal
- Comparing two trees

Program

```
#include  <iostream>
using namespace std ;

int main ( )
{
  btree bt ;

  bt.insert ( 10 ) ;
  bt.insert ( 6 ) ;
  bt.insert ( 8 ) ;
  bt.insert ( 15 ) ;
  bt.insert ( 20 ) ;

  bt.traverse ( ) ;
}
```

class btree

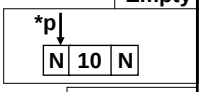
```
class btree
{
  private :
    struct btnode {
      btnode *lc ;
      int data ;
      btnode *rc ;
    } *root ;

  public :
    btree( ) ;
    void insert ( int num ) ;
    static void insert ( btnode **sr, int num ) ;
    void traverse ( ) ;
    static void inorder ( btnode *sr  ) ;
    static void preorder ( btnode *sr ) ;
    static void postorder ( btnode *sr ) ;
    static void del ( btnode *sr ) ;
    ~btree( ) ;
} ;
```

```
btree :: btree( )
{
  root = NULL ;
}

void btree :: insert ( int num )
{
  insert ( &root, num ) ;
}
```

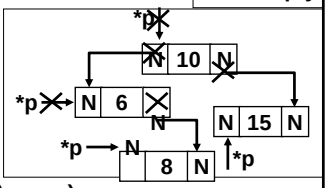
insert

```
void btree :: insert ( btnode  **p, int  num )
{
    if ( *p == NULL )
    {
        *p = new btnode ;
        ( *p ) -> lc = NULL ;
        ( *p ) -> data = num ;
        ( *p ) -> rc = NULL ;
    }
    else
    {
        if ( num < ( *p ) -> data )
            insert ( &( ( *p ) -> lc ), num ) ;
        else
            insert ( &( ( *p ) -> rc ), num ) ;
    }
}
```

Empty

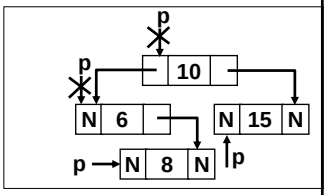


Non-empty



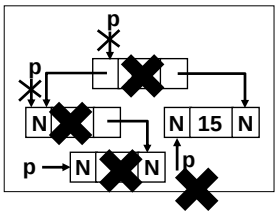
inorder, preorder

```
void btree :: inorder ( btnode  *p )
{
    if ( p != NULL )
    {
        inorder ( p -> lc ) ;
        cout << "\t" << p -> data ;
        inorder ( p -> rc ) ;
    }
}

void
btree :: preorder ( btnode  *p )
{
    if ( p != NULL ) {
        cout << "\t" << p -> data ;
        preorder ( p -> lc ) ;
        preorder ( p -> rc ) ;
    }
}
```



postorder

```
void btree :: postorder ( btnode  *p )
{
    if ( p != NULL )
    {
        postorder ( p -> lc ) ;
        postorder ( p -> rc ) ;
        cout << "\t" << p -> data ;
    }
}
void btree :: del ( btnode  *p)
{
    if ( p != NULL )
    {
        del ( p -> lc ) ;
        del ( p -> rc ) ;
        delete p ;
    }
}
```




```
btree :: ~btree ( )
{
  del ( root ) ;
}
```

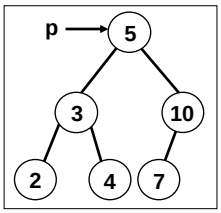
Compare Trees

```
#include  <iostream>
using namespace std ;

int main( )
{
    btree  bt1, bt2 ;

    bt1.insert ( 5 ) ;
    bt1.insert ( 3 ) ;
    bt1.insert ( 10 ) ;
    bt1.insert ( 4 ) ;
    bt1.insert ( 2 ) ;

    bt2.insert ( 5 ) ;
    bt2.insert ( 3 ) ;
    bt2.insert ( 10 ) ;
    bt2.insert ( 4 ) ;
    bt2.insert ( 2 ) ;
```

```
  bool i ;
  btree :: compare (
              bt1, bt2, i ) ;
  if ( i == true )
      cout << "Equal" << endl ;
  else
      cout ( "Unequal" << endl ;
}
void btree :: compare (
    btree& bt1 , btree& bt2 ,
    bool& flag ) {
  compare ( bt1.root , bt2.root,
      flag ) ;
}
```

Compare func.

```
void btree :: compare ( btnode *p, btnode  *q, bool&  flag )
{
    flag = false ;
    if ( p == NULL && q == NULL )
        flag = true ;
    if ( p != NULL && q != NULL )
    {
        if ( p -> data != q -> data )
            flag = false ;
        else
        {
            compare ( p -> lc, q -> lc, flag ) ;
            if ( flag != false )
                compare ( p -> rc, q -> rc, flag ) ;
        }
    }
}
```



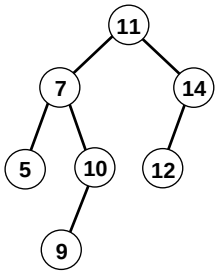
Binary Trees

Asang Dani
asang@ksetindia.com

Objectives

- How to perform insertion on binary trees
- How to delete a node from a binary tree
- How to search a node in a binary tree

Successor And Predecessor



In - 5, 7, 9, 10, 11, 12, 14
Pre - 11, 7, 5, 10, 9, 14, 12
Post - 5, 9, 10, 7, 12, 14, 11

- In - order successor of 7 is 9
- In - order predecessor of 7 is 5

- Pre - order successor of 5 is 10
- Pre - order predecessor of 5 is 7

- Post - order successor of 12 is 14
- Post - order predecessor of 12 is 7

In - order successor of a node with two child is always a leaf node or a node with only right child

Program

```
#include  <iostream>
using namespace std ;

int main( )
{
    btree  bt ;

    bt.insert ( 11 ) ;
    bt.insert ( 7 ) ;
    bt.insert ( 10 ) ;
    bt.insert ( 14 ) ;
    bt.insert ( 5 ) ;
    bt.insert ( 9 ) ;
    bt.insert ( 12 ) ;
    cout << "\nBinary Tree: ";
    bt.display ( ) ;

    bt.remove ( 14 ) ;
    bt.remove ( 7 ) ;
    bt.remove ( 5 ) ;

    cout << "\nBinary Tree: "  ;
    bt.display ( ) ;
}
```

class btree

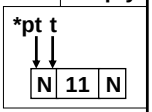
```
class btree {
 private :
   struct btnode {
     btnode *lc ;
     int data ;
     btnode *rc ;
   } *root ;
   static void del ( btnode *sr ) ;

 public :
   btree( ) ;
      void insert ( int num ) ;
      void remove ( int num ) ;
      void display( ) ;
   static void insert ( btnode **sr, int ) ;
   static bool search ( btnode **sr, int n, btnode **par, btnode **x ) ;
   static void rem ( btnode **sr, int ) ;
   static void inorder ( btnode *sr ) ;
      ~btree( ) ;
} ;
```

```
btree :: btree( )
{
  root = NULL ;
}

void btree :: insert ( int num )
{
  insert ( &root, num ) ;
}
```

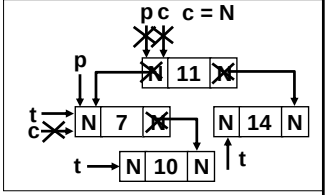
insert

```
void btree :: insert ( btnode  **pt, int  num )
{
    btnode  *t , *p, *c ;

    t = new  btnode ;

    t -> data = num ;
    t -> rc =t -> lc = NULL ;
    if ( *pt == NULL )
        *pt = t ;
    else
    {
        p = c = *pt ;
        while ( c != NULL )
        {
            p = c ;
            num < p -> data ? ( c = p -> lc ) : ( c = p -> rc ) ;
        }
        num < p -> data ? ( p -> lc = t ) : ( p -> rc = t ) ;
    }
}
```

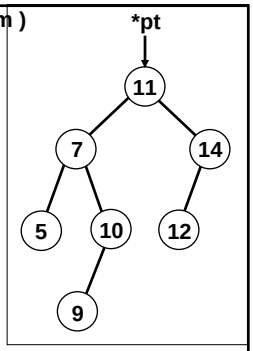
Empty



Non-empty



```
void btree :: rem ( btnode  **pt, int  num )
{
    btnode  *p, *c, *csucc ;
    bool found ;

    if ( *pt == NULL )
    {
        cout << "\nTree is empty" ;
        return ;
    }
    p = c = NULL ;
    found = search ( pt, num, &p, &c ) ;

    if ( found == false )
    {
        cout << "\nData not found" ;
        return ;
    }
```

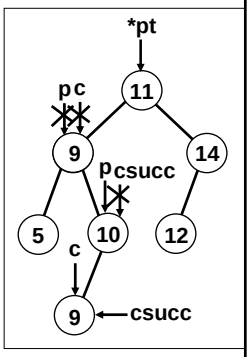


rem function

Contd...

...Contd

```
    if ( c -> lc != NULL && c -> rc != NULL )
    {
        p = c ;
        csucc = c -> rc ;
        while ( csucc -> lc != NULL )
        {
            p = csucc ;
            csucc = csucc -> lc ;
        }

        c -> data = csucc -> data ;
        c = csucc ;
    }
```



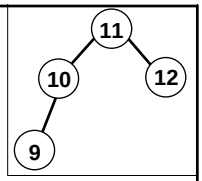
Contd...

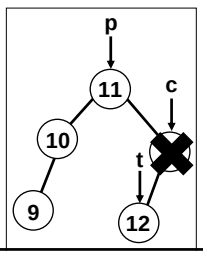
...Contd

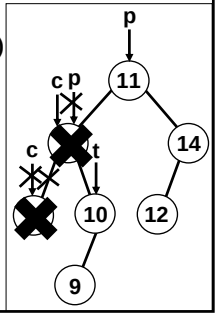
```
    if ( c -> lc == NULL && c -> rc == NULL )
        t = NULL ;
```

btnode *t

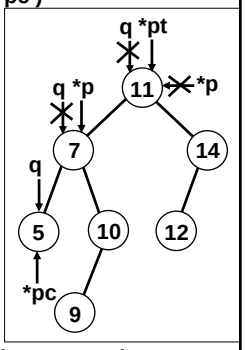
```
    if ( c -> lc == NULL && c -> rc != NULL )
        t = c -> rc ;

    if ( c -> lc != NULL && c -> rc == NULL )
        t = c -> lc ;

    if ( p -> lc == c )
        p -> lc = t ;
    else
        p -> rc = t ;

    delete c ;
}
```







```
bool btree :: search ( btnode  **pt, int  num, btnode  **p,
                       btnode  **pc )
{
    btnode  *q ;
    q = *pt ;
    *p = NULL ;
    while ( q != NULL )
    {
        if ( q -> data == num )
        {
            *pc = q ;
            return  true ;
        }
        *p = q ;
        num < q -> data ? ( q = q -> lc ) : ( q = q -> rc ) ;
    }
    return  false ;
}
```



search function

remove, etc.

```
void btree :: remove ( int num )
{
  rem ( &root, num ) ;
}

void btree :: display( )
{
  inorder ( root ) ;
}

void btree :: inorder ( btnode *sr )
{
  if ( sr != NULL )
  {
    inorder ( sr -> lc ) ;
    cout << sr -> data << "\t" ;
    inorder ( sr -> rc ) ;
  }
}
```

```
btree :: ~btree( )
{
  del ( root ) ;
}

void btree :: del ( btnode *sr )
{
  if ( sr != NULL )
  {
    del ( sr -> lc ) ;
    del ( sr -> rc ) ;
  }
  delete sr ;
}
```

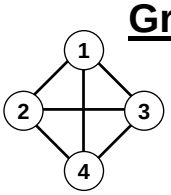
Graphs

Yashavant Kanetkar

Objectives

- What are graphs
- The terminology associated with graphs
- Representation of graphs
- Adjacency lists

Graphs

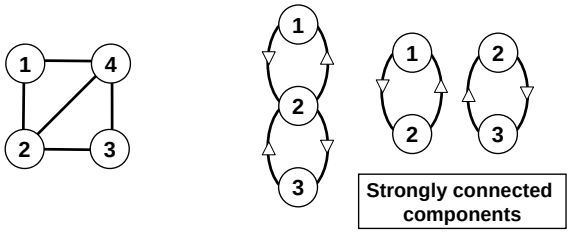


Undirected Graph

Digraph

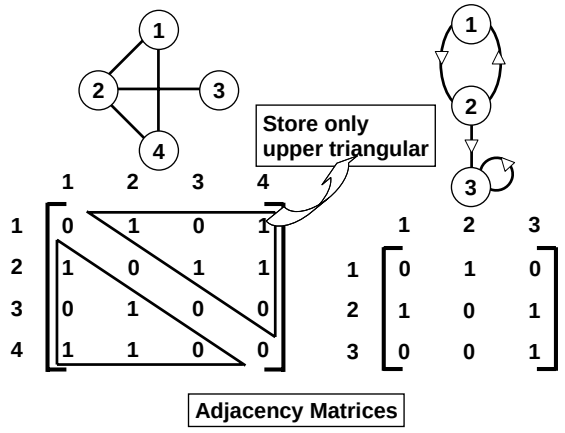
- Graph is set of v & e
  v - vertices (finite & non-empty), e - edges (pair of vertices)
- Undirected graph - edge is unordered, (1, 2) & (2, 1) are same
- Digraph - edge is ordered, <1, 2> & <2, 1> are different
- In <1, 2> 1 is tail and 2 is head
- In undirected graph 1 and 2 are adjacent
- In undirected graph edge (1, 2) is incident on 1 & 2
- In digraph 2 is adjacent to 3, while 3 is adjacent from 2
- In digraph edges <1, 2> and <2, 1> are incident to 1 & 2

More Terminology

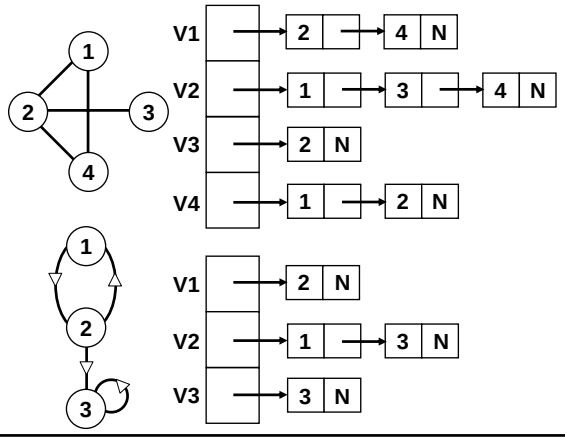


- Path - from 1 to 3 is sequence of vertices 1, 4, 3 with edges (1, 4), (4, 3)
- Simple path - starting & ending vertex distinct. 1, 4, 3
- Cyclic path - starting & ending vertex is same. 1, 2, 4, 1
- Connected - if there is a path from v1 to v2
- Strongly connected - if there is a path from v1 to v2 and v2 to v1.

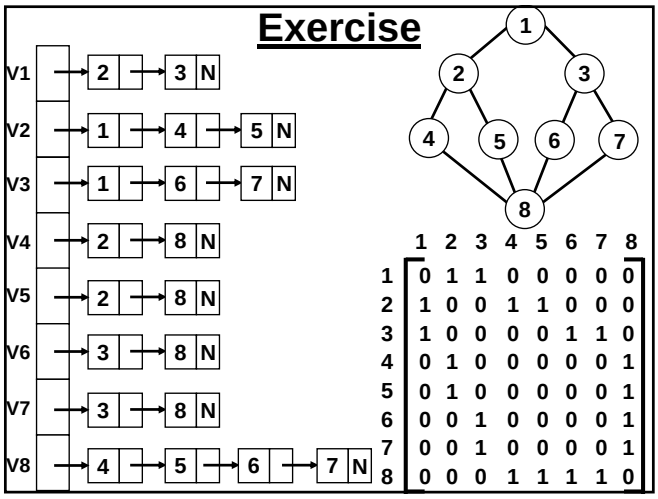
Graph Representation



Adjacency List



Exercise



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	0	1	1	0	0	0
3	1	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	1
5	0	1	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	1	0	0	0	0	1
8	0	0	0	1	1	1	1	0

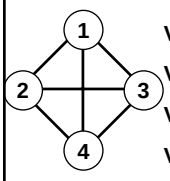
Adjacency Multilist

Yashavant Kanetkar

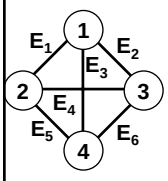
Objectives

- Adjacency matrices
- Adjacency multilists
- Orthogonal representation of graphs

Adjacency Multilist



V1	2	3	4	N
V2	1	3	4	N
V3	1	2	4	N
V4	1	2	3	N



	m	V_i	V_j	Next link for V_i	Next link for V_j
E_1		1	2	E_2	E_4
E_2		1	3	E_3	E_4
E_3		1	4	N	E_5
E_4		2	3	E_5	E_6
E_5		2	4	N	E_6
E_6		3	4	N	N

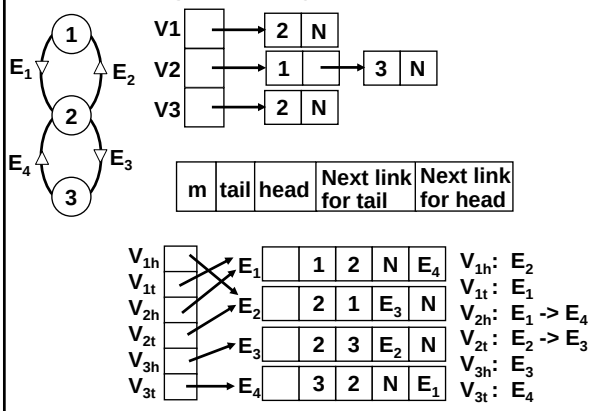
V_1: E_1-> E_2 -> E_3

V_2: E_1-> E_4 -> E_5

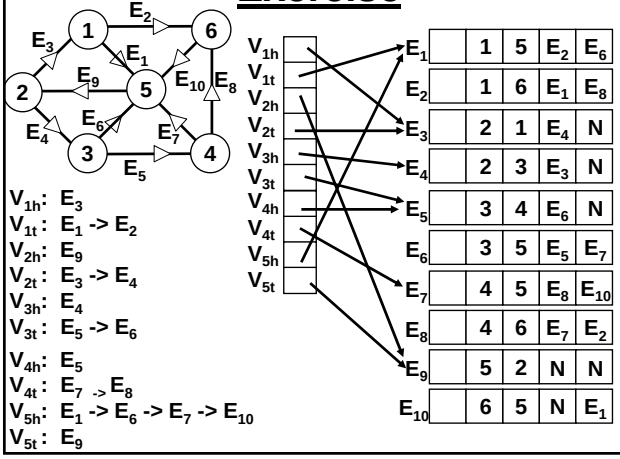
V_3: E_2-> E_4 -> E_6

V_4: E_3-> E_5 -> E_6

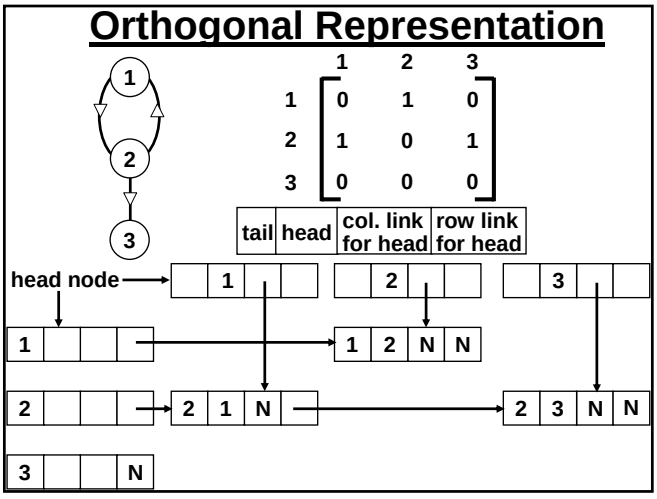
Adjacency Multilist



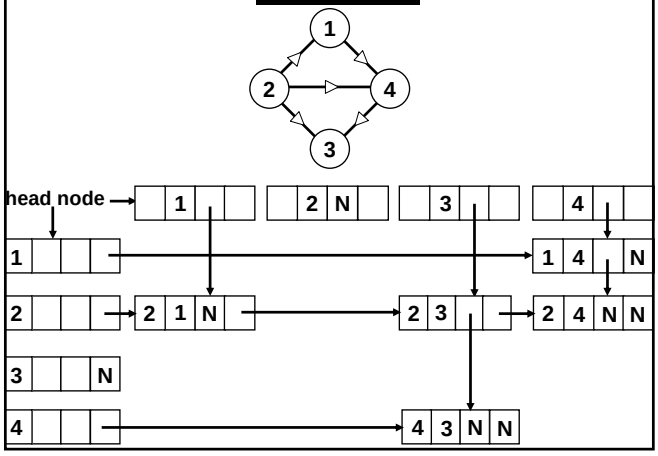
Exercise



Orthogonal Representation

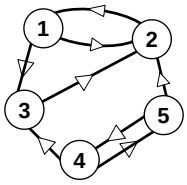


Exercise



Exercise

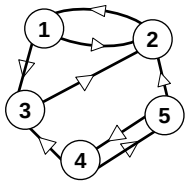
Give all graphical representations for the following graph

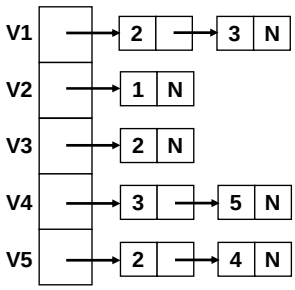


	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	1	0	1	0

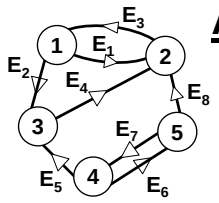
Adjacency Matrices

Adjacency List

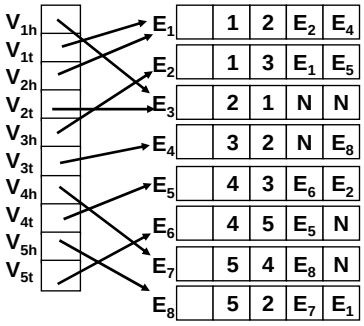




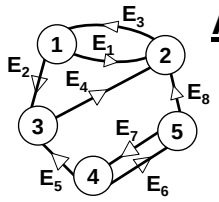
Adjacency Multilist



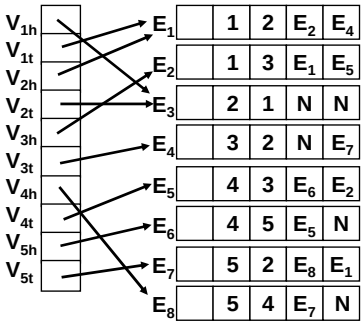
V_{1h}: E_3
V_{1t}: E_1 -> E_2
V_{2h}: E_1 -> E_4 -> E_8
V_{2t}: E_3
V_{3h}: E_2 -> E_5
V_{3t}: E_4
V_{4h}: E_7
V_{4t}: E_5 - > E_6
V_{5h}: E_6
V_{5t}: E_7 -> E_8



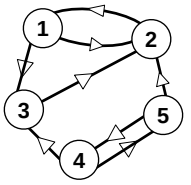
Adjacency Multilist



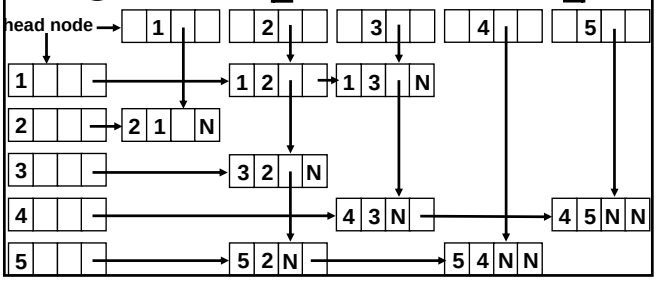
V_{1h}: E_3
V_{1t}: E_1 -> E_2
V_{2h}: E_1 -> E_4 -> E_7
V_{2t}: E_3
V_{3h}: E_2 -> E_5
V_{3t}: E_4
V_{4h}: E_8
V_{4t}: E_5 - > E_6
V_{5h}: E_6
V_{5t}: E_7 -> E_8



Orthogonal



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	1	0	1	0



Depth First Search

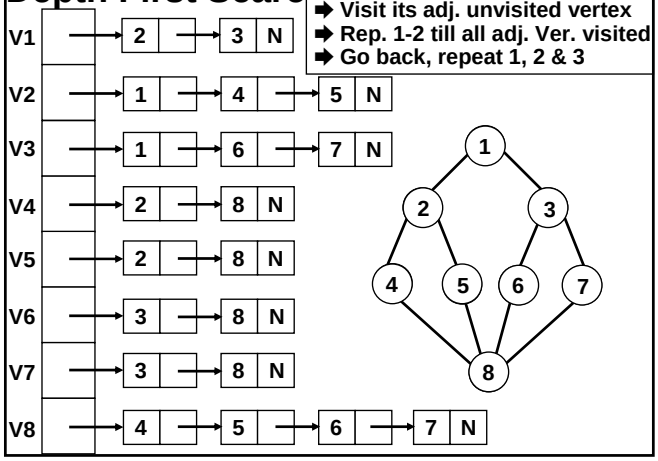
Asang Dani
asang@ksetindia.com

Objectives

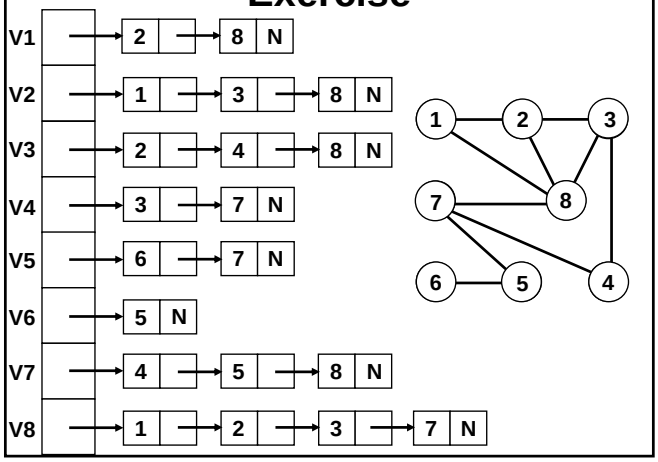
- Depth First Search Algorithm
- Depth First Search Example
- Depth First Search Program

Depth First Searc

- Visit a vertex, mark it visited
- Visit its adj. unvisited vertex
- Rep. 1-2 till all adj. Ver. visited
- Go back, repeat 1, 2 & 3



Exercise



Program

```
# include <iostream>
using namespae std ;

int main( )
{
     node  *arr [ MAX ] ;
     node  *v1, *v2, *v3 ;
     graph g ;
     v1 = g.add ( 2 ) ;
     v2 = g.add ( 3 ) ;
     arr[ 0 ] = v1 ;
     v1 -> next = v2 ;

     v1 = g.add ( 1 ) ;
     v2 = g.add ( 4 ) ;
     v3 = g.add ( 5 ) ;
     arr[ 1 ] = v1 ;
     v1 -> next = v2 ;
     v2 -> next = v3 ;

     v1 = g.add ( 1 ) ;
     v2 = g.add ( 6 ) ;
     v3 = g.add ( 7 ) ;
     arr[ 2 ] = v1 ;
     v1 -> next = v2 ;
     v2 -> next = v3 ;

     v1 = g.add ( 2 ) ;
     v2 = g.add ( 8 ) ;
     arr[ 3 ] = v1 ;
     v1 -> next = v2 ;
```

Contd...

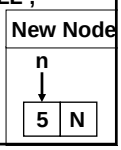
...Contd

```
     v1 = g.add ( 2 ) ;
     v2 = g.add ( 8 ) ;
     arr[ 4 ] = v1 ;
     v1 -> next = v2 ;

     v1 = g.add ( 3 ) ;
     v2 = g.add ( 8 ) ;
     arr[ 5 ] = v1 ;
     v1 -> next = v2 ;

     v1 = g.add ( 3 ) ;
     v2 = g.add ( 8 ) ;
     arr[ 6 ] = v1 ;
     v1 -> next = v2 ;

     v1 = g.add ( 4 ) ;
     v2 = g.add ( 5 ) ;
     v3 = g.add ( 6 ) ;
     v4 = g.add ( 7 ) ;

     arr[ 7 ] = v1 ;
     v1 -> next = v2 ;
     v2 -> next = v3 ;
     v3 -> next = v4 ;
node  *v4
     g.dfs ( 1, arr ) ;
     for ( i = 0 ; i < MAX ; i++ )
          g.del ( arr[ i ] ) ;
}
                    int  i
```

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

class graph
{
      private :
        bool visited [ MAX ] ;
      public :
        graph ( ) ;
        void dfs ( int v, node **p ) ;
        node* add ( int val ) ;
        void del ( node *n ) ;
} ;
```

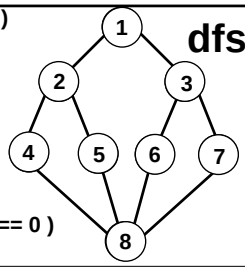
class graph

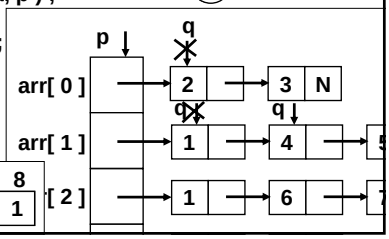
```
graph :: graph( )
{
  for ( int i = 0 ; i < MAX ; i++ )
    visited [ i ] = false ;
}
node * graph :: add ( int  val )
{
      node  *n ;
      n = new node ;
      n -> data = val ;
      n -> next = NULL ;
      return  n ;
}
```




```
void graph :: dfs ( int  v, node **p )
{
      struct  node  *q ;
      visited [ v - 1 ] = true ;
      cout << v << "\t" ;
      q = * ( p + v - 1 ) ;
      while ( q != NULL )
      {
            if ( visited [ q -> data - 1 ] == 0 )
                  dfs ( q -> data, p ) ;
            else
                  q = q -> next ;
      }
}
```

dfs ( )



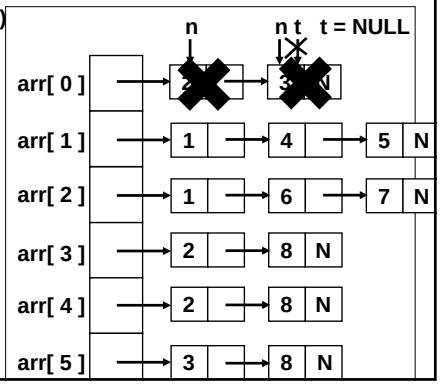


1	2	4	8	5	6	3	7

1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1

```
void graph :: del ( node  *n )
{
  node  *t ;
  while ( n != NULL )
  {
     t = n -> next ;
     delete  n  ;
     n = t ;
  }
}
```

del( )



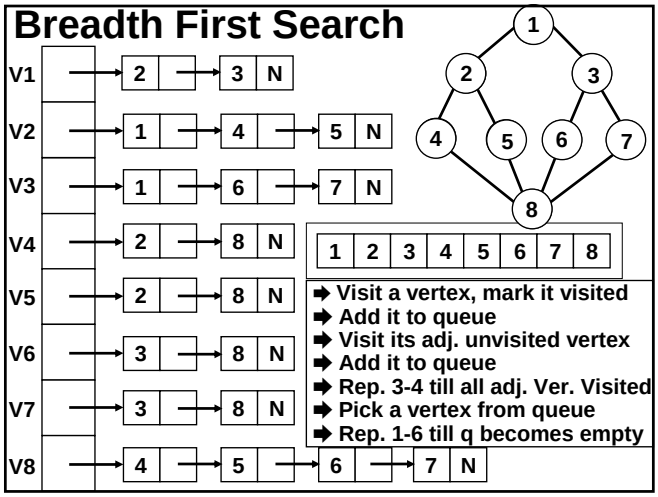
Breadth First Search

Asang Dani
asang@ksetindia.com

Objectives

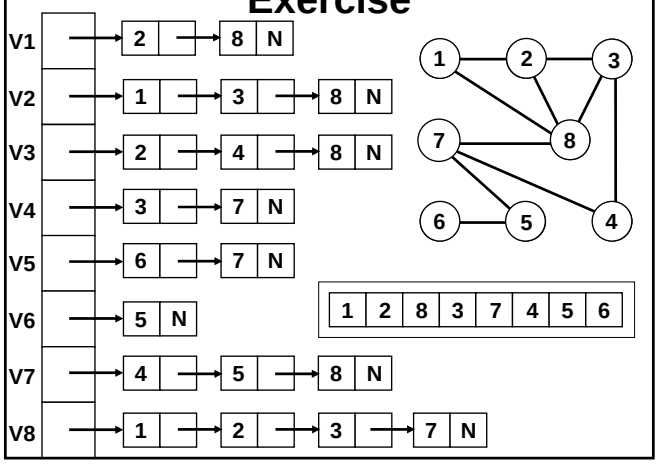
- Breadth First Search Algorithm
- Breadth First Search Example
- Breadth First Search Program

Breadth First Search



- Visit a vertex, mark it visited
- Add it to queue
- Visit its adj. unvisited vertex
- Add it to queue
- Rep. 3-4 till all adj. Ver. Visited
- Pick a vertex from queue
- Rep. 1-6 till q becomes empty

Exercise



class graph

```
#include <iostream>
#include <cstdlib>

using namespace std ;

const int MAX = 8 ;
struct node
{
  int data ;
  node *next ;
} ;

class graph
{
  private :
    bool visited [ MAX ] ;
    int q [ MAX ] ;
    int f, r ;
  public :
    graph( ) ;
    void bfs ( int v, node **p ) ;
    node * add ( int val ) ;
    void addqueue ( int v ) ;
    int delqueue ( ) ;
    static bool isempty ( ) ;
    void del ( node *n ) ;
} ;

  graph :: graph( )
  {
    for ( int i = 0 ; i < MAX ; i++ )
      visited [ i ] = false ;
    f = r = -1 ;
  }
```

bfs

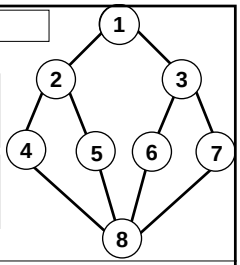
```
void graph :: bfs ( int  v, node  **p )
{
    node *u ;

    visited[ v - 1 ] = true ;
    cout << v << "\t" ;
    addqueue ( v ) ;

    while ( isempty ( ) == false )
    {
        v = delqueue ( ) ;
        u = * ( p + v - 1 ) ;
```

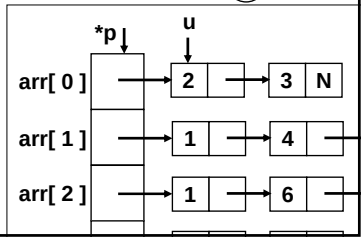
1

1	2	3	4
1	0	0	0

queue

1			





Contd...

...Contd

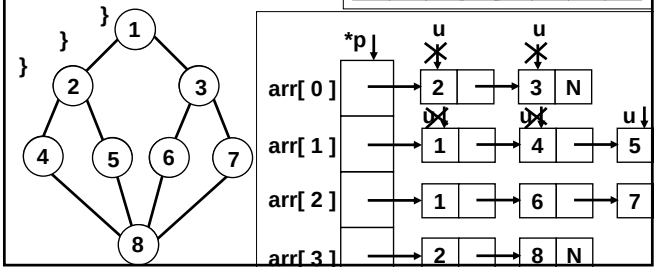
```
while ( u != NULL )
{
    if ( ! visited [ u -> data - 1 ] )
    {
        addqueue ( u -> data ) ;
        visited [ u -> data - 1 ] = true ;
        cout << u->data << "\t" ;
    }
    u = u -> next ;
}
```

1	2	3	4	5	6	7	8

queue

2	3	4	5

1	1	1	1	1	1	1	1



addqueue( )

```
addqueue ( int vertex )
{
    if ( r == MAX - 1 )
    {
        cout << "\Full."  ;
        exit( ) ;
    }

    r + + ;
    q [ r ] = vertex ;

    if ( f == - 1 )
      f = 0 ;
}
```

delqueue( ) & isempty( )

```
int graph :: delqueue ( )
{
    int  data ;
    if ( f == - 1 )
    {
        cout << "\nEmpty." ;
        exit( ) ;
    }
    data = q [ f ] ;
    if ( f == r )
      f = r = -1 ;
    else
      f + + ;
    return  data ;
}
```

```
bool graph :: isempty ( )
{
    if ( f == - 1 )
        return  true ;
    return  false ;
}
node* graph :: add ( int val )
{
  node *newnode = new node ;
  newnode -> data = val ;
  return newnode ;
}
```

Spanning Tree

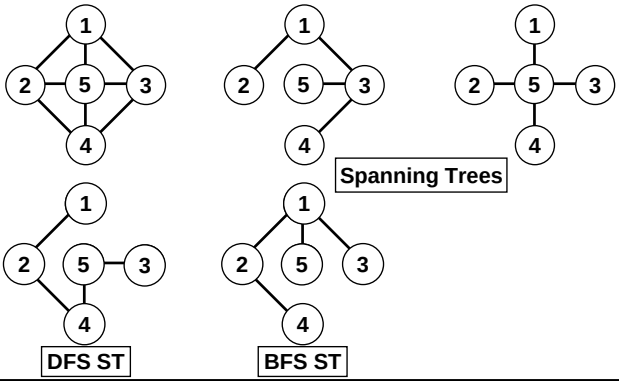
Asang Dani

Objectives

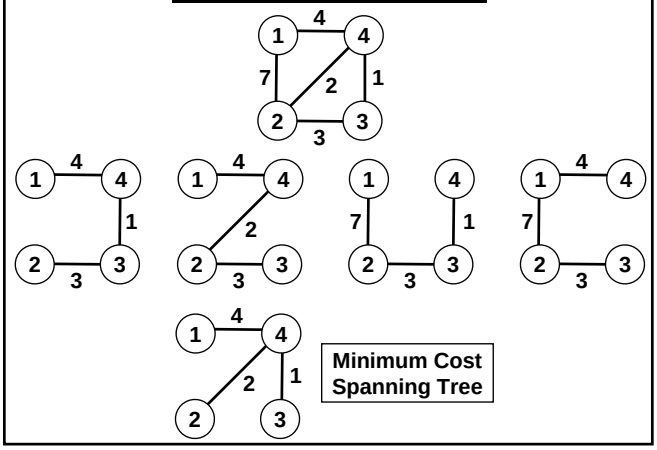
- Spanning Tree
- Cost of Spanning Tree
- Kruskal's Algorithm to determine minimum cost Spanning Tree

Spanning Tree

- Undirected tree
- Contains edges necessary to connect all vertices

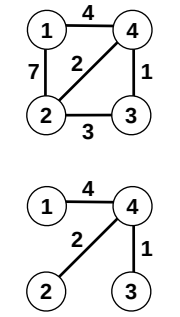


Calculating Cost



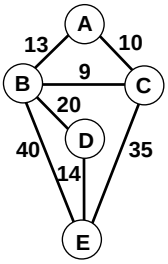
Kruskal's Algo

- Insert edges in inc. order of cost
- Reject if it forms a cyclic path



Edge	Cost	Tree	Action
4 - 3	1		Included
4 - 2	2		Included
3 - 2	3		Rejected
4 - 1	4		Included

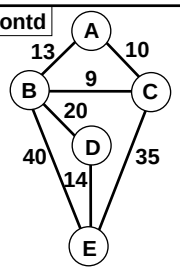
Exercise

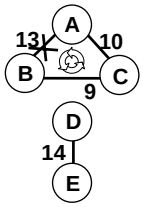
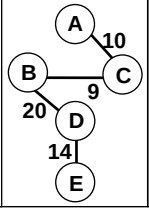


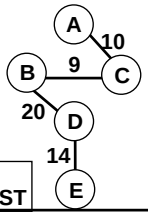
Edge	Cost	Tree	Action
B - C	9		Included
A - C	10		Included

Contd...

...Contd



Edge	Cost	Tree	Action
A - B	13		Rejected
E - D	14		Included
B - D	20		Included



Min. Cost ST

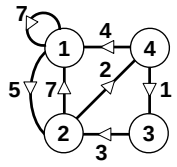
Dijkstra's Algorithm

Asang Dani

Objectives

- Dijkstra's Algorithm to determine minimum cost Spanning Tree
- AOV Network
- Exercise

Dijkstra's Algorithm

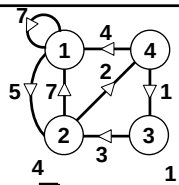


	1	2	3	4
1	7	5	∞	∞
2	7	∞	∞	2
3	∞	3	∞	∞
4	4	∞	1	∞

	1	2	3	4
1	11	12	-	-
2	21	-	-	24
3	-	32	-	-
4	41	-	43	-

Contd...

...Contd



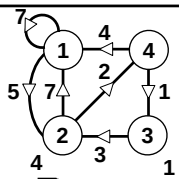
	1	2	3	4
1	7	5	∞	∞
2	7	∞	∞	2
3	∞	3	∞	∞
4	4	∞	1	∞

	1	2	3	4
1	11	12	-	-
2	21	-	-	24
3	-	32	-	-
4	41	-	43	-

	1	2	3	4
1	7	5	∞	∞
2	7	12	∞	2
3	∞	3	∞	∞
4	4	9	1	∞

	1	2	3	4
1	11	12	-	-
2	21	212	-	24
3	-	32	-	-
4	41	412	43	-

...Contd



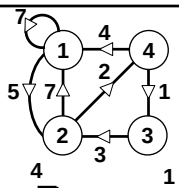
	1	2	3	4
1	7	5	∞	∞
2	7	12	∞	2
3	∞	3	∞	∞
4	4	9	1	∞

	1	2	3	4
1	11	12	-	-
2	21	212	-	24
3	-	32	-	-
4	41	412	43	-

	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	9	1	11

	1	2	3	4
1	11	12	-	124
2	21	212	-	24
3	321	32	-	324
4	41	412	43	4124

...Contd



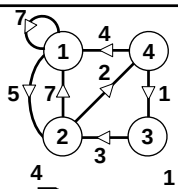
	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	9	1	11

	1	2	3	4
1	11	12	-	124
2	21	212	-	24
3	321	32	-	324
4	41	412	43	4124

	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	4	1	6

	1	2	3	4
1	11	12	-	124
2	21	212	-	24
3	321	32	-	324
4	41	432	43	4324

...Contd



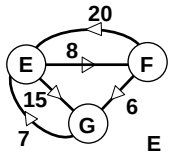
	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	4	1	6

	1	2	3	4
1	11	12	-	124
2	21	212	-	24
3	321	32	-	324
4	41	432	43	4324

	1	2	3	4
1	7	5	8	7
2	6	6	3	2
3	9	3	6	5
4	4	4	1	6

	1	2	3	4
1	11	12	1243	124
2	241	2432	243	24
3	3241	32	3243	324
4	41	432	43	4324

Exercise



	E	F	G
E	∞	8	15
F	20	∞	6
G	7	∞	∞

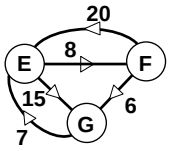
	E	F	G
E	-	EF	EG
F	FE	-	FG
G	GE	-	-

	E	F	G
E	∞	8	15
F	20	28	6
G	7	15	22

	E	F	G
E	-	EF	EG
F	FE	FEF	FG
G	GE	GEF	GEG

Contd...

...Contd



	E	F	G
E	∞	8	15
F	20	28	6
G	7	15	22

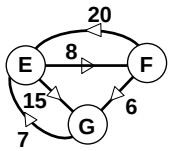
	E	F	G
E	-	EF	EG
F	FE	FEF	FG
G	GE	GEF	GEG

	E	F	G
E	28	8	14
F	20	28	6
G	7	15	21

	E	F	G
E	EFE	EF	EFG
F	FE	FEF	FG
G	GE	GEF	GEFG

Contd...

...Contd



	E	F	G
E	28	8	14
F	20	28	6
G	7	15	21

	E	F	G
E	EFE	EF	EFG
F	FE	FEF	FG
G	GE	GEF	GEFG

	E	F	G
E	21	8	14
F	13	21	6
G	7	15	21

	E	F	G
E	EFGE	EF	EFG
F	FGE	FGEF	FG
G	GE	GEF	GEFG

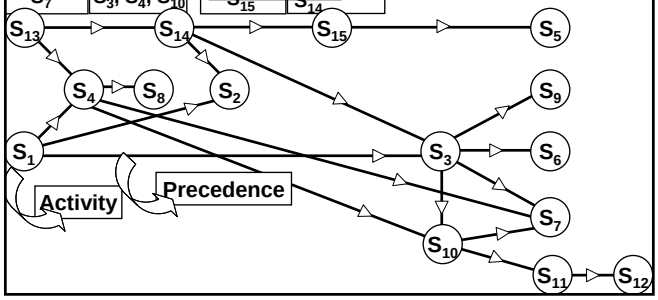
Sub. No.	Prereq.
~~S_1~~	~~None~~
~~S_2~~	~~S_1, S_{14}~~
~~S_3~~	~~S_1, S_{14}~~
~~S_4~~	~~S_1, S_{13}~~
~~S_5~~	~~S_{15}~~
~~S_6~~	~~S_3~~
~~S_7~~	~~S_3, S_4, S_{10}~~

Sub. No.	Prereq.
~~S_8~~	~~S_4~~
~~S_9~~	~~S_3~~
~~S_{10}~~	~~S_3, S_4~~
~~S_{11}~~	~~S_{10}~~
~~S_{12}~~	~~S_{11}~~
~~S_{13}~~	~~None~~
~~S_{14}~~	~~S_{13}~~
~~S_{15}~~	~~S_{14}~~

AOV N/W

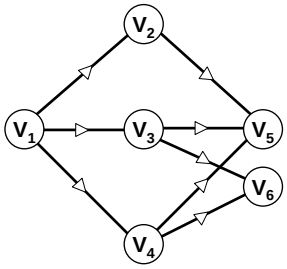
Activity On Vertex Network

V - Activity / Task
E - Precedence Relation



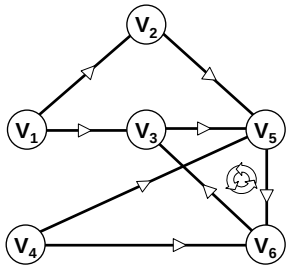
Exercise - I

Vertex	Prerequisite
~~V_1~~	~~None~~
~~V_2~~	~~V_1~~
~~V_3~~	~~V_1~~
~~V_4~~	~~V_1~~
~~V_5~~	~~V_2, V_3, V_4~~
~~V_6~~	~~V_3, V_4~~



Exercise - II

Vertex	Prerequisite
~~V_1~~	~~None~~
~~V_2~~	~~V_1~~
V_3	V_1, V_6
~~V_4~~	~~None~~
V_5	V_2, V_3, V_4
V_6	V_4, V_5



Not Feasible