Data Structures Through C

Student Handout Book

Contents

Lecture 7: Sorting

- Selection sort and its analysis
- Bubble sort and its analysis
- Radix sort

Lecture 8: Sorting and Recursion

- Insertion sort
- Recursive functions

Lecture 9: Quick Sort

- Splitting mechanism
- Quick sort algorithm
- Quick sort program

Lecture 10: Structures

- Defining and using structures
- Array of structures
- Copying structure variables
- Nested structures
- Passing structure elements
- Passing structures

Lecture 11: Structures and Polynomials

- Polynomial representation
- Passing array of structures
- General operations on polynomials

Lecture 12: Two Dimensional Arrays

- Using two dimensional arrays
- Saddle point of an array

Lecture 13: Sparse Matrices

- Sparse matrices
- Passing arrays
- 3 tuple conversion

Lecture 14: Transpose of Sparse Matrices

- Transpose of a sparse matrix

Lecture 15: Addition of Sparse Matrices

- Addition of sparse matrices

Lecture 16: Multiplication of Sparse Matrices

- Matrix multiplication

Lecture 25: Doubly Linked Lists

- Doubly linked lists
- Common operations on a doubly linked list

Lecture 26: Sparse Matrices as Linked Lists I

- Representing sparse matrices in the form of linked lists

Lecture 27: Sparse Matrices as Linked Lists II

- Add new node to linked Lists
- Display contents of Linked List
- Delete all nodes in the Linked List
- Program to implement sparse matrices as linked lists

Lecture 28: Recursive LL

- Recursive **count( )**
- Recursive **compare( )**
- Recursive **copy( )**
- Recursive **append( )**

Lecture 29: Linked Lists Using Unions

- What are **union**s
- Declaring **union**s
- Using **union**s
- Generalized linked lists

Lecture 30: Generalized Linked Lists

- Copying generalized linked lists
- Depth of generalized linked lists
- Comparing generalized linked lists

Lecture 31: Stacks

- Stack and its utilities
- Program to implement stack as an array

Lecture 32: Stack as a Linked List

- Program to implement stack as a linked list

Lecture 33: Stack Expressions

- Different form of expressions

Lecture 34: Stack Operations

- Infix to Postfix conversion

Lecture 35: Postfix

- Evaluation of Postfix form

Lecture 36: Infix To Prefix

- Infix to Prefix conversion

Lecture 37: Postfix To Prefix

- Evaluation of Prefix form
- Postfix to Prefix conversion

Lecture 38: Postfix To Infix

- Postfix to Infix conversion

Lecture 39: More Conversions

- Prefix to Postfix conversion
- Prefix to Infix conversion

Lecture 40: Queue I

- Queue
- Program to implement queue as an array

Lecture 41: Queue II

- Limitations of queue
- Circular queue

Lecture 42: Deque and Priority Queue

- Deque
- Program to implement deque
- Priority queue

Lecture 43: Trees

- What are trees
- Tree terminology
- Strictly and complete trees

Lecture 44: Tree Traversal

- Preorder, Postorder and Inorder traversal
- Reconstruction of trees

Lecture 45: Binary Search Trees I

- Representing trees using arrays
- Linked representation of trees
- Binary search trees

Lecture 46: Binary Search Trees II

- Traversing of trees using Inorder, Preorder and Postorder
- Non recursive Inorder traversal of trees
- Non recursive Preorder traversal of trees

- Non recursive Postorder traversal of trees
- Comparison of two trees

Lecture 47: Binary Tree

- Successor and predecessor
- Insertion of a node in a binary search tree
- Deleting an existing node from a binary search tree
- Searching a node in a binary search tree

Lecture 48: Threaded Binary Tree

- Threaded binary trees
- Representation of threaded binary trees
- Inserting a node in a threaded binary tree
- Traversing the threaded binary tree in inorder

Lecture 49: Heap

- Heap
- Creation of a heap
- Heap sort

Lecture 50: AVL Trees and B Tree

- Program to create a heap
- AVL tree
- 2 - 3 tree
- B tree

Lecture 51: Graphs

- Graphs
- Terminology associated with graphs
- Representation of graphs
- Adjacency list

Lecture 52: Adjacency Multilist

- Adjacency matrices
- Adjacency multilist
- Orthogonal representation of graphs

Lecture 53: Depth First Search

- Depth first search
- Program to implement depth first search algorithm

Lecture 54: Breadth First Search

- Breadth first search

- Program to implement breadth first search algorithm

Lecture 55: Spanning Tree

- What is a spanning tree
- Cost of spanning tree
- Kruskal's algorithm to determine minimum cost spanning tree

Lecture 56: Dijkstra's Algorithm

- Dijkstra's algorithm to determine minimum cost spanning tree
- AOV network

Lecture 57: Memory Management I

- Memory management
- First fit algorithm
- Best fit algorithm

Lecture 58: Memory Management II

- Next fit algorithm

Lecture 59: Garbage Collection

- Garbage Collection
- Marking algorithm
- Compaction

Lecture 60: File Organization

- Files
- File Organization

C Revision - I

Yashavant Kanetkar

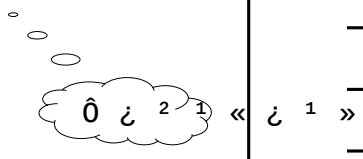
Objectives

- The importance of data structures
- The data types in C
- What are instructions and its types
- The decision control instruction
- Working of logical operators
- Different types of loops
- Difference between break, continue and exit
- Case control instruction

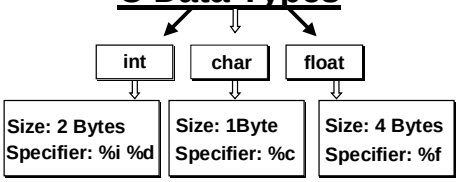
Data Structures

- What are they

  Way of organizing data and opns. performed on it
- Where are they used
  - Car Parking
  - File Storage
  - Dictionary
  - Shortest route
  - Searching Law Books
  - Sorting
  - Device Drivers
  - Evaluation of expression
  - Networking



C Data Types



C Instructions

- Type Declaration Instructions
  int i, j ; char ch, dh ; float a, b ;
- Arithmetic Instructions
  c = 5 / 9.0 * (f - 32) ;
  + - * / % - Arithmetic operators
- Input / Output Instructions
  printf( ), scanf( )
- Control Instructions
  Control the sequence of execution of instructions
  Types - Sequence - Default
  - Decision
  - Repetition
  - Case

Decision Control Instruction

- Implementation - if-else, ? :
- General Form

```
if ( condition )
   statement1 ;
else
   statement2 ;
```

- Cond. is usually built using <, >, <=, >=, = =, !=
  && || !
- else block is optional
- Condition – True – Replaced by 1
- Condition – False – Replaced by 0
- Truth in C is nonzero
- Falsity is zero

Relational Operators

Logical Operators

Working of && And ||

cond1	cond2	cond1 && cond2	cond1 || cond2
True	True	True	True
False	False	False	False
True	False	False	True
False	True	False	True

cond1 && cond2

cond2 executed only if cond1 is true

cond1 || cond2

cond2 executed only if cond1 is false

Repetition Control Instruction

```
for ( i = 1; i <= 10 ; i++ )
   printf ( "Hi" ) ;
```

```
i = 1 ;
while ( i <= 10 )
{
   printf ( "Hi" ) ;
   i++ ;
}
```

```
i = 1 ;
do
{
    printf ( "Hi" ) ;
    i++ ;
} while ( i <= 10 ) ;
```

Tips

- for - Finite no. of times
- while - Unknown no. of times
- do - while - At least once
- Loop counters can be int, long int, float or char
- Loop counters can be incremented or decremented by any suitable step value

Effects of *break* & *continue*

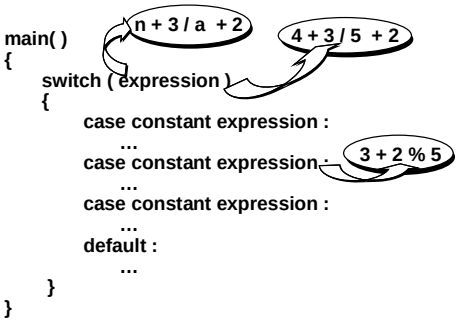
```
while ( .... )
{
  if ( cond1 )
     break ;
  if ( cond2 )
     continue ;
}
```

```
for ( .. ; .. ; .. )
{
  if ( cond1 )
     break ;
  if ( cond2 )
     continue ;
}
```

```
do
{
  if ( cond1 )
     break ;
  if ( cond2 )
     continue ;
} while ( .... ) ;
```

break - Terminates Loop
exit( ) - Terminates Program

Case Control Instruction




```
main( )
{
    switch ( expression )
    {
        case constant expression :
            ...
        case constant expression :
            ...
        case constant expression :
            ...
        default :
            ...
    }
}
```

C Revision - II

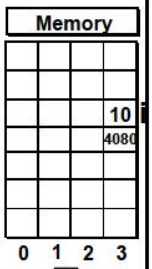
Yashavant Kanetkar

Objectives

- Pointers and their usage
- Function call - by value and by reference
- Arrays and their working

Pointers

```
main( )
{
    int  i = 10 ;
    printf ( "\n value of  i = %d" , i ) ;
    printf ( " \n address of  i  = %d" ,  &i ) ;
    printf ( " \n value of  i  = %d" ,  *( &i ) ) ;
}
```



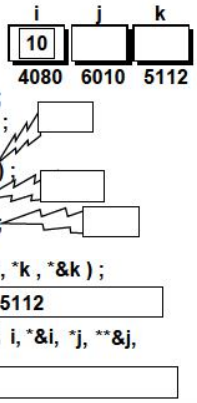
Pointer Operators

& - 'Address Of' Operator
* - 'Value at Address' Operator
- 'Indirection' Operator

Address
Reference
Memory location
Cell number

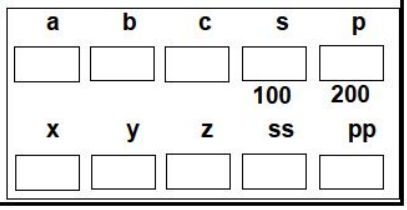
The Next Step

```
main( )
{
    int  i = 10 ;  int  *j ; int **k ;
    printf ( "\n value of i = %d" , i ) ;
    printf ( " \n address of i = %d " , &i ) ;
    printf ( " \n value of i = %d " , *( &i ) ) ;
    j = &i ;
    printf ( " \n address of  j = %d " , &j ) ;
    printf ( " \n value of  j = %d " , j  ) ;
    printf ( " \n value of  j = %d " , *&j  ) ;
    k = &j ;
    printf  ( " \n %d %d %d %d " , k ,  &k , *k , *&k ) ;
    printf  ( " \n  %d %d %d %d %d %d ",  i, *&i,  *j,  **&j,
                **k,  ***&k  ) ;
}
```



Types Of Calls

```
main( )
{
    int  a = 10, b = 20, c = 30, s, p ;
    sumprod ( a, b, c, &s, &p ) ;
    printf ( "%d%d", s, p ) ;
}
sumprod ( int x, int y, int z , int  *ss, int  *pp )
{
    *ss =  x + y + z ;
    *pp =  x * y * z ;
}
```



Arrays

```
main( )
{
    int  m1, m2, m3, per ;   int  i ;
    for ( i = 1 ; i <= 10 ; i++ )
    {
        printf ( "Enter Marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per = ( m1 + m2 + m3 ) / 3 ;
        printf ( "%d", per ) ;
    }
    printf ( "%d", per ) ;
}
```

- Use 10 variables each holding 1 value
- Use 1 variable holding all 10 values → Array
- Array - Variable that can hold more than 1 value at a time

```
main( )
{
    int  m1, m2, m3, per[ 10 ] ;
    int  i ;

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

    for ( i = 0 ; i <= 9 ; i++ )
        printf ( "%d", per[ i ] ) ;
}
```

Array?

```
    0
0   A
```

Screen

Array Subtleties

```
main( )
{
   int  a[ ] = { 7, 6, 11, -2, 26 } ;
   int  b[ 10 ] ;
   int  c[10 ] = { 16, 13, -8, -7, 25 } ;
   printf ( "%d%d", sizeof ( a ), sizeof ( b ) ) ;
   printf ( "%d%d", a[ 0 ], b[ 0 ] ) ;
   scanf ( "%d%d%d", &c[ 7 ], &c[ 8 ], &c[ 9 ] ) ;
   c[ 5 ] = 3 + 7 % 2 ;
   c[ 6 ] = c[ 1 ] + c[ 3 ] / 16 ;
}
```

optional

compulsory

- ➔ Arrays can be initialized
- ➔ Arrays have storage classes
- ➔ Array elements can be scanned
- ➔ Array elements can be calculated
- ➔ Arithmetic on array elements is allowed

Storage

```
main( )
{
   int  i = 3,  j = 20, k = -5, l = 7, m = 11 ;
   int  a[ ] = { 3, 20, -5, 7, 11 } ;  int  ii ;
   printf ( "%u %u %u %u %u", &i, &j, &k, &l, &m ) ;
   for ( ii = 0 ; ii <= 4 ; ii++ )
         printf ( "%u", &a[ ii ] ) ;
}
```

i	j	k	l	m
100	400	500	700	600

```
int  a[ ] = { 2, 1.4, 'A', 6 } ;
```

- Adjacency
- Similarity

a[0]	a[1]	a[2]	a[3]	a[4]
502	504	506	508	510

Searching

Yashavant Kanetkar

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

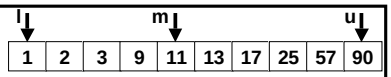
Array Operations

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

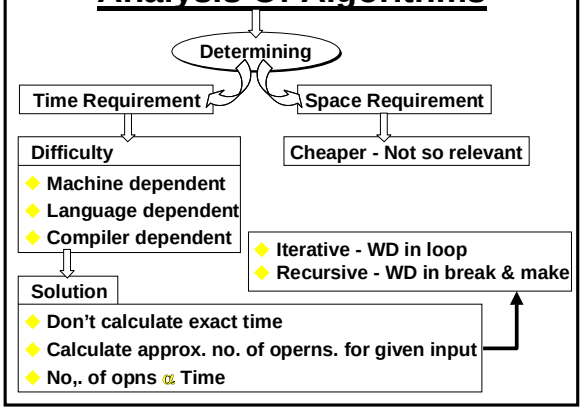
Searching & Frequency Count

Yashavant Kanetkar

Objectives

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

Analysis Of Algorithms



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

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.

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.

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 who 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	
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

Analysis Of Searching Methods

Yashavant Kanetkar

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

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

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

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

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

(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

Yashavant Kanetkar

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	6748	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

Yashavant Kanetkar

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 ] ) ;
}
```

									i	i+1
17	6	13	12	2						
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

Yashavant Kanetkar

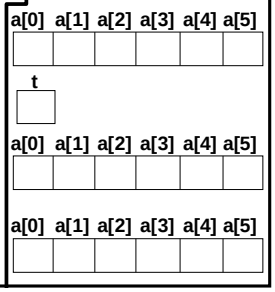
Objectives

- 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 ] ) ;
}
```



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 ) ;
}
```

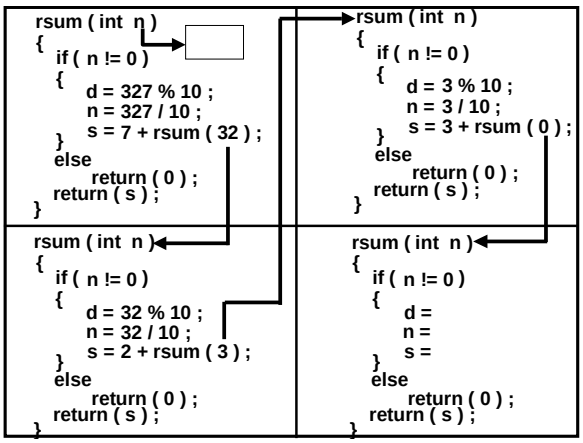
31698
d5

485
d3

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 ) ;
}
```

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

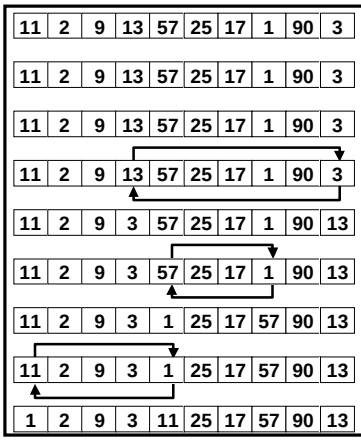
Quick Sort

Yashavant Kanetkar

Objectives

- Splitting mechanism in quick sort
- Quick sort algorithm
- Quick sort 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…

..Contd

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

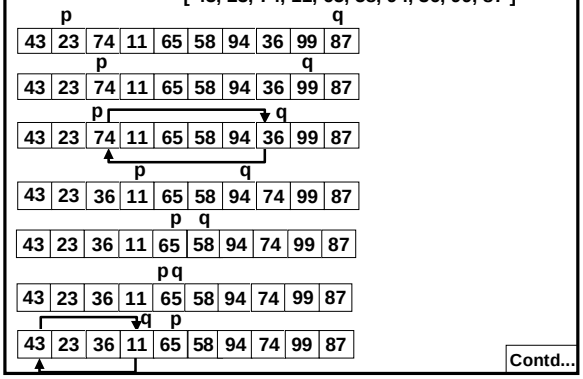
Problem

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

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

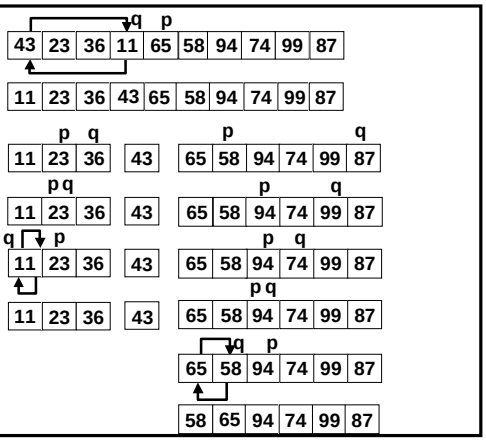
Exercise

Wrtie 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

Yashavant Kanetkar

Objectives

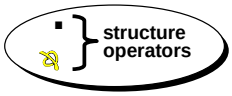
- Defining and using structures
- Creating an array of structures
- How to copy one structure variable into another
- Using nested structures
- Passing structure elements
- Passing structures

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

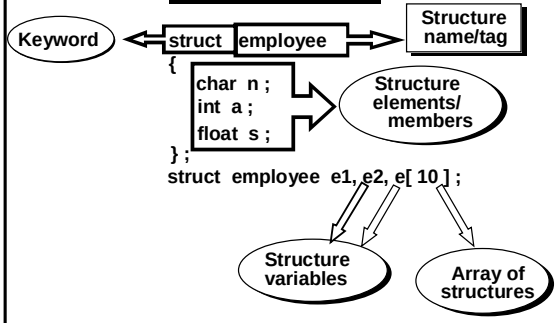
```
main( )
{
    struct  employee
    {
        char  n ;
        int  a ;
        float  s ;
    } ;
    struct  employee  e1 = { 'A', 23, 4000.50 } ;
    struct  employee  e2 = { 'X', 27, 5000.00 } ;
    struct  employee  e3 = { 'Y', 28, 6000.75 } ;
    printf ( "%c %d %f", e1.n, e1.a, e1.s ) ;
 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

struct employee *z[3] ;
```

Ptr. to structure

Ptr. to array of structures

402

408

422

Array of Ptrs to structures

Copying

e1

Rahul	23	400.50

e2

Rahul	23	400.50

e3

Rahul	23	400.50

```
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 ) ;
}
```

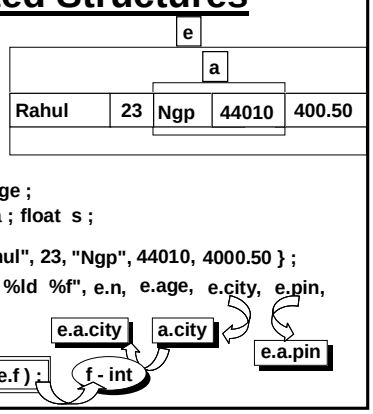
piecemeal copying

strcpy (e2.n, e1.n) ;

copying at one shot

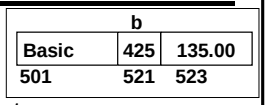
Nested Structures

```
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 ) ;
}
```



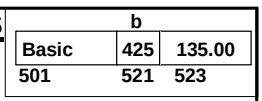
Passing Structure Elements

```
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 ) ;
}
```



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 ) ;
}
```



Structures & Polynomials

Yashavant Kanetkar

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 Addⁿ

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

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

```
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 ]  = pa[ i ] ;
      {
        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 ;
}
```

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 ;
}
```

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

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

Solution

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

Two Dimensional Arrays

Yashavant Kanetkar

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

optional

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

compulsory

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

b[4][4]

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

c[4][5]

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

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

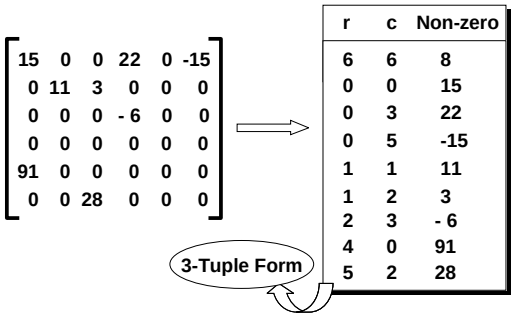
Sparse Matrices

Yashavant Kanetkar

Objectives

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

Sparse Matrices



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, 4, 5 ) ;
    fun2 ( a, 4, 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

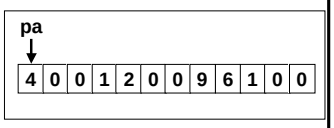
4	0	0	1	2	0	0	9	6	1	0	0

p

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

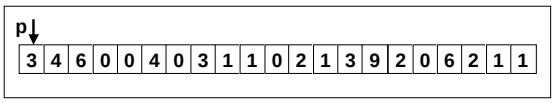
Cont…

```
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

Yashavant Kanetkar

Objectives

- 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 give sparse matrix is same as the original sparse matrix.

Addition of Sparse Matrices

Yashavant Kanetkar

Objectives

➔ How to 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
..	..	..				

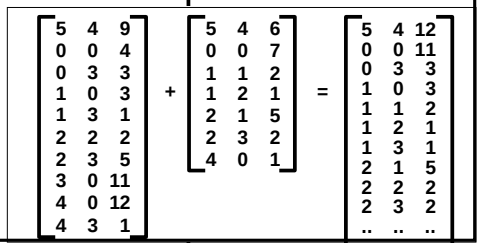
```
int * add ( int  *s1, int  *s2 )
{
  int *p,  i, j, k ;
  int  max, maxa, maxb ;
  int  rowa, cola, vala ;
  int  rowb, colb, valb ;

  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++ ;
}
```

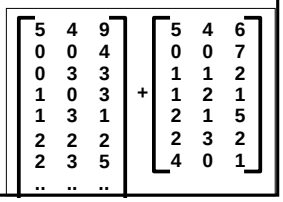
```
if ( rowa > rowb )
{
    p[ k * 3 + 0 ] = rowb ;
    p[ k * 3 + 1 ] = colb ;
    p[ k * 3 + 2 ] = valb ;
    j++ ;
}
```



Cont...

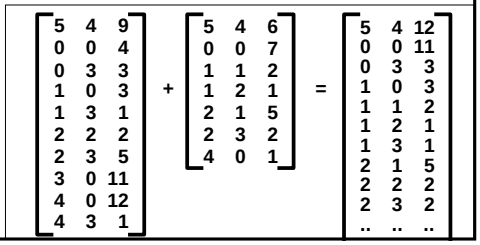
```
if ( rowa == rowb )
{
    if ( cola == colb )
    {
      p[ k*3+0 ] = rowa ;
      p[ k* 3+1 ] = cola ;
      p[ k*3+2 ] = vala + valb ;
      i++ ; j++ ; max- - ;
    }

    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

Yashavant Kanetkar

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_x ( 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

Yashavant Kanetkar

Objectives

- How two dimensional arrays are stored
- 3-D arrays
- 4-Darrays
- 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} →

int a[m][n]

a_{ij} →

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] ;

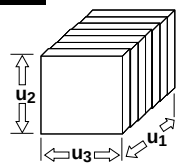
a_{23} →

int a[m][n]

a_{ij} →

3-D Array

a[1…u_1, 1…u_2, 1…u_3]



Row major

a[i][j][k]

Column major

a[i][j][k]

4-D Array

a[1…u_1, 1…u_2, 1…u_3, 1…u_4]

Row major

a[i][j][k][l]

Column major

a[i][j][k][l]

n-Dimensional Array

a[1…u_1, 1…u_2, 1…u_3, …, u_n]

Row major

a[i_1][i_2][i_3][..][i_n]

α + (i_1 – 1) * $u_2u_3u_4…u_n$
+ (i_2 – 1) * $u_3u_4…u_n$
+ (i_3 – 1) * $u_4u_5…u_n$
+ (i_4 – 1) * $u_5u_6…u_n$
+
.
.
.
+ (i_{n-1} – 1) * u_n
+ (i_n – 1)

Column major

a[i_1][i_2][i_3][..][i_n]

α + (i_1 – 1) * $u_2u_3u_4…u_n$
+ (i_2 – 1) * $u_3u_4…u_n$
+ (i_3 – 1) * $u_4u_5…u_n$
+ (i_4 – 1) * $u_5u_6…u_n$
+
.
.
.
+ (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
```

Problem

Find the location of an element A[7][8][2][5] relative to the first element of the array A[6:8][7:10][2:5][5:8]

Dynamic Memory Allocation

Yashavant Kanetkar

Objectives

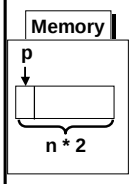
- Limitations on arrays
- Dynamic memory allocation
- How the memory allocation is done

Arrays

No Flexibility

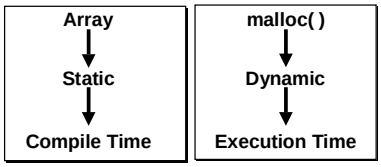
```
main( )
{
    int  m1, m2, m3, i, per [ 10 ] ;
    for ( i = 0 ; i <= 9 ; i++ )
    {
        printf ( "Enter marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per [ i ] = ( m1 + m2 + m3 ) / 3 ;
        printf ( "%d", per [ i ]) ;
    }
}
```

Dynamic Allocation




```
# include "alloc.h"
main( )
{
    int  *p ;  int  m1, m2, m3, i, per ;
    printf ( "Enter no. of students" ) ;
    scanf ( "%d", &n ) ;
    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



- 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)

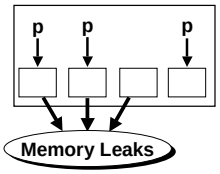
Loader

Fatal

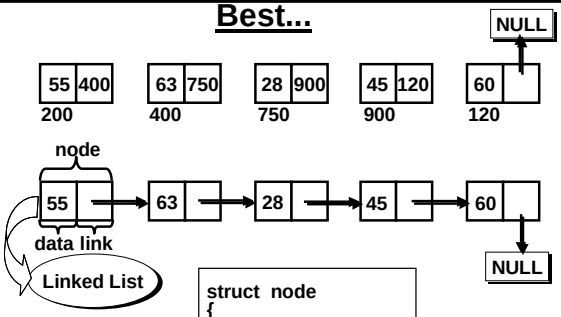
- If too many globals - Loader fails
- If too many locals - Stack overflow
- If too much dynamic demand - Heap full

Better Still...

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



Best...




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

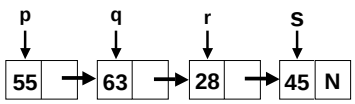
Linked List

Yashavant Kanetkar

Objectives

- What are memory leaks
- What are linked lists
- How to create a linked list of records

Linked List

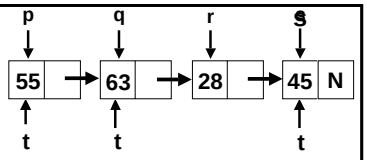



```
main( )
{
   struct  node
   {
       int  data ;
       struct  node  *link ;
   } ;
  struct node *p, *q, *r, *s ;
   malloc ( sizeof ( struct node ) ) ;
  q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
  r =  . . .
  s =  . . .
  p ⌫ data = 55 ;q ⌫ data = 63 ;r ⌫ data = 28 ;s ⌫ data = 45
  p ⌫ link = q ;  q ⌫ link = r ;r ⌫ link = s ; s ⌫ link = NULL ;
```

Cont...

..Cont

```
# include "alloc.h"
main( )
{
   ...
   ...
   ...
   ...
   printf ( "%d %d %d %d",  p ⌫ data, q ⌫ data,
                            r ⌫ data, s ⌫ data ) ;
   printf ( "%d %d %d",  p ⌫ data, p ⌫ link ⌫ data,
                         p ⌫ link ⌫ link ⌫ data ) ;
   t = p ;
   while ( t != NULL )
   {
       printf  ( "%d", t ⌫ data ) ; t = t ⌫ link ;
   }
}
```

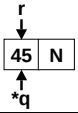


Most General

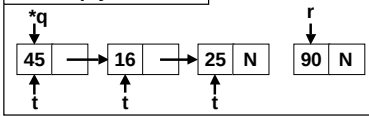
```
# include "alloc.h"
struct  node
{
    int  per ; struct  node  *link ;
} ;
main( )
{
    struct  node  *p ;
    char  ch = 'Y' ; int  pp, m1, m2, m3 ;
    p = NULL ;
    while ( ch == 'Y' )
    {
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        pp = ( m1 + m2 + m3 ) / 3 ;
        add ( &p, pp ) ;
        printf ( "Another student y/n" ) ;
        ch = getche( ) ;
    }
}
```

```
add ( struct node **q, int  pp )
{
   struct  node  *r ;  struct  node  *t ;
   r = ( struct node * ) malloc ( sizeof ( struct 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 ;
   }
}
```

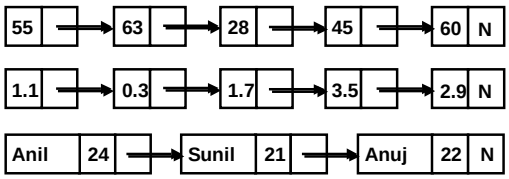
Empty linked list



Non-empty linked list



Variety...




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

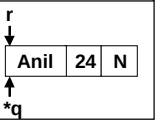
Linked List Of Records

```
# include "alloc.h"
struct  node
{
   char  name [ 20 ] ; int  age ; struct  node  *link ;
} ;
main( )
{
    struct  node  *p ;
    struct  node  t ;  char  ch = 'Y' ;
    p = NULL ;
    while ( ch == 'Y' )
    {
        printf ( "\nEnter name & age: " ) ;
        scanf ( "%s%d", t.name, &t.age ) ;
        add (  &p, t ) ;
        printf ( "Another student Y/N" ) ;
        ch = getche( ) ;
    }
}
```

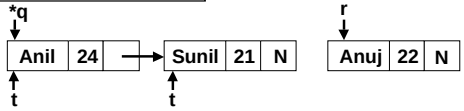
Contd...

```
add ( struct node **q, struct  node  n )
{
   struct  node  *r ;  struct  node  *t ;
   r = ( struct node * ) malloc ( sizeof ( struct  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 - I

Yashavant Kanetkar

Objectives

- How to perform various operations on linked lists
- How to append a new node to a link list
- How to add a new node at the beginning of a linked list
- How to add a new node at a specified position in a linked list
- How to delete a node from a linked list
- How to delete all nodes in the list

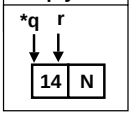
LL Operations

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct node  *p ;
    p = NULL ;
    int  c ;
    append ( &p, 14 ) ;
    append ( &p, 30 ) ;
    append ( &p, 25 ) ;
    append ( &p, 17 ) ;
    display ( p ) ;
    addatbeg ( &p, 7 ) ;
    addatbeg ( &p, 58 ) ;
    display ( p ) ;
    addafter ( p, 7, 0 ) ;
    addafter ( p, 2, 1 ) ;
    addafter ( p, 5, 99 ) ;
    display ( p ) ;
    c = count ( p ) ;
    printf ( "\nNo. of eles. %d", c ) ;
    del ( &p, 30 ) ;
    del ( &p, 10 ) ;
    display ( p ) ;
    c = count ( p ) ;
    printf ( "\nNo. of eles. %d", c ) ;
    deleteall ( &p ) ;
}
```

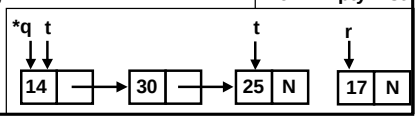
Append

```
append ( struct node  **q, int  n )
{
   struct  node  *r, *t ;
   r = ( struct  node * ) malloc ( sizeof ( struct node ) ) ;
   r -> data = n ;
   r -> link = NULL ;
   if ( *q == NULL )
      *q = r ;
   else
   {
      t = *q ;
      while ( t -> link != NULL )
         t = t -> link ;
      t -> link = r ;
   }
}
```

Empty List

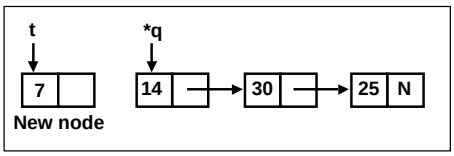


Non-Empty List

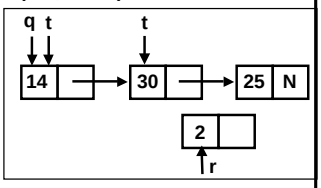


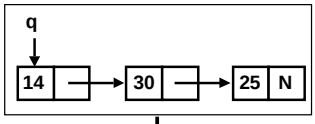
Add At Beginning

```
addatbeg ( struct  node   **q, int  n )
{
   struct node  *t ;
   t = ( struct  node * ) malloc ( sizeof ( struct node ) ) ;
   t -> data = n ;
   t -> link = *q ;
   *q = t ;
}
```




```
addafter ( struct node  *q, int  pos, int  n )
{
   struct  node  *t, *r ;
   int  i ;
   t = q ;
   for ( i = 0 ; i < pos ; i++ )
   {
      if ( t -> link == NULL )
         break ;
      t = t -> link ;
   }
   r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
   r -> data = n ;
   r -> link = t -> link ;
   t -> link = r ;
}
```



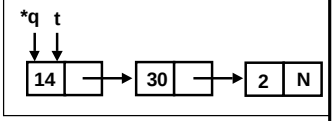



```
display ( struct  node  *q )
{
   while ( q != NULL )
   {
      printf ( "%d\t", q->data ) ;
      q = q -> link ;
   }
}
```

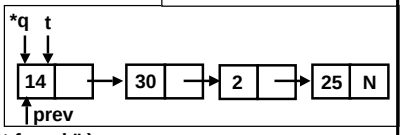
```
int count ( struct  node  *q )
{
    int c = 0 ;
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }
    return c ;
}
```

```
del ( struct node  **q, int  n )
{
   struct  node  *t , *prev ;
   t = *q ;
   while ( t != NULL )
   {
      if ( t -> data == n )
      {
         if ( t == *q )
            *q = t -> link ;
         else
            prev -> link = t -> link ;
         free ( t ) ;
         return ;
      }
      prev = t ;
      t = t -> link ;
   }
   printf ( "\nEle. not found." ) ;
}
```

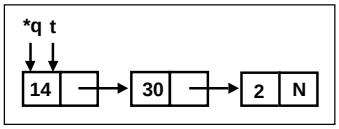
Node to be Deleted = 14



Node to be Deleted = 2



Delete All Nodes




```
deleteall ( struct node  **q )
{
    struct node  *t ;
    while ( *q != NULL )
    {
        t = *q ;
        *q = ( *q ) -> link ;
        free ( t ) ;
    }
}
```

Operations On Linked List II

Yashavant Kanetkar

Objectives

- How to arrange the linked list elements in ascending order
- How to sorting the elements in the linked list

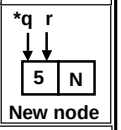
Ascending Order LL

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct node  *p ;  int  c ;
    p = NULL ;
    add ( &p, 5 ) ;
    add ( &p, 1 ) ;
    add ( &p, 6 ) ;
    add ( &p, 4 ) ;
    add ( &p, 7 ) ;
    display ( p ) ;
    c = count ( p ) ;
    printf ( "\nNo. of eles. %d", c ) ;
}
```

add( ) Function

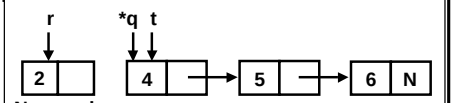
```
add ( struct node  **q, int  n )
{
   struct node  *r, *t ;
   t = *q ;
   r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
   r -> data = n ;
   if ( *q == NULL || ( *q ) -> data > n )
   {
      *q = r ;
     ( *q ) -> link = t ;
   }
```

Empty LL



New node

Addition At Beginning

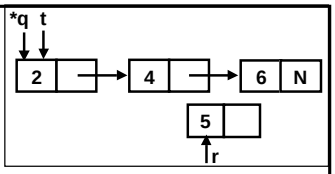


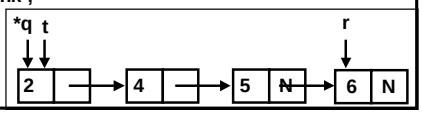
New node

Cont...

Cont...

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

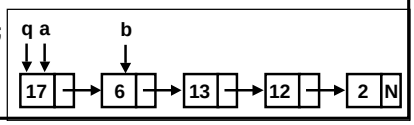




Sorting

```
#include "alloc.h"
struct node
{
   int data ;
   struct node *link ;
} ;
main( )
{
   struct node *p = NULL ;
   append ( &p, 17 ) ;
   append ( &p, 6 ) ;
   append ( &p, 13 ) ;
   append ( &p, 12 ) ;
   append ( &p, 2 ) ;
   display ( p ) ;
   ssort ( p ) ;
   display ( p ) ;
   deleteall ( p ) ;
}
```

```
ssort ( struct node  *q )
{
   struct  node *a, *b ;
   int  n, i,  j, t ;
    a = q ;   n = count ( a ) ;
   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 ;
   }
}
```



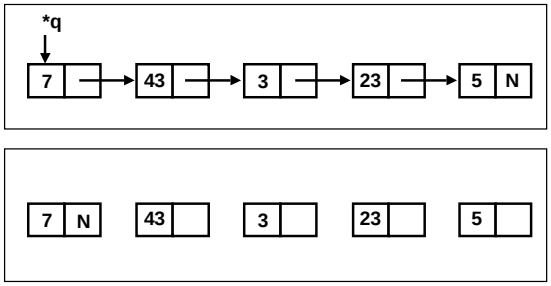
Operations On Linked List - III

Yashavant Kanetkar

Objectives

- How to perform various operations on linked lists
- How to reversing a linked list
- How to merge two linked lists

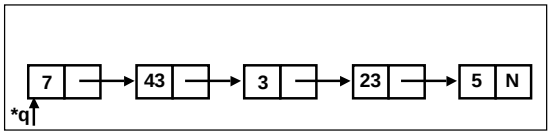
Reversing LL



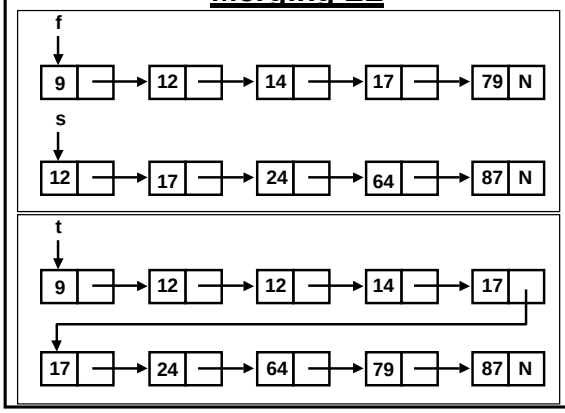
Reversing LL

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct  node  *p = NULL ;
    append ( &p, 7 ) ;
    append ( &p, 43 ) ;
    append ( &p, 3 ) ;
    append ( &p, 23 ) ;
    append ( &p, 5 ) ;
    display ( p ) ;
    reverse ( &p ) ;
    display ( p ) ;
    deleteall ( p ) ;
}
```

```
reverse ( struct  node  **q )
{
    struct  node  *r, *prev, *t ;
    r = *q ;   prev = NULL ;
    while ( r != NULL )
    {
        t = prev ;
        prev = r ;
        r = r -> link ;
        prev -> link = t ;
    }
    *q = prev ;
}
```



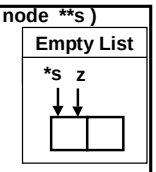
Merging LL

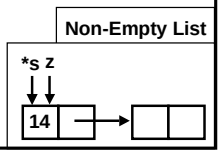


Merging LLs

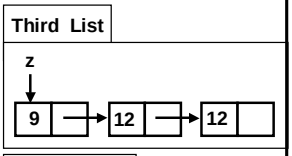
```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct  node  *f, *s, *t ;
    f = s = t = NULL ;
    add ( &f, 9 ) ;
    add ( &f, 12 ) ;
    add ( &f, 14 ) ;
    add ( &f, 17 ) ;
    add ( &f, 79 ) ;
    printf ( "First  LL: " ) ;
    display ( f ) ;
    add ( &s, 12 ) ;
    add ( &s, 17 ) ;
    add ( &s, 24 ) ;
    add ( &s, 64 ) ;
    add ( &s, 87 ) ;
    printf ( "\nSecond LL: " ) ;
    display ( s ) ;
    merge ( f, s, &t ) ;
    printf ( "\nMerged LL: " ) ;
    display ( t ) ;
    deleteall ( f ) ;
    deleteall ( s ) ;
    deleteall ( t ) ;
}
```

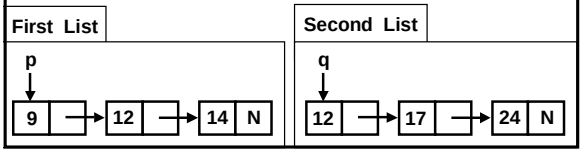
```
merge ( struct node  *p, struct node  *q, struct node  **s )
{
    struct node  *z ;
    while ( p != NULL && q != NULL )
    {
        if ( *s == NULL )
            z = *s = ( struct node * ) malloc (
                                    sizeof ( struct node ) ) ;
        else
        {
            z -> link = ( struct node *) malloc (
                                    sizeof ( struct node ) ) ;
            z = z -> link ;
        }
```



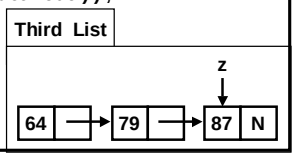



```
        if ( p -> data <= q -> data )
        {
            z -> data = p -> data ;
            p = p -> link ;
        }
        else
        {
            z -> data = q -> data ;
            q = q -> link ;
        }
    } // while
```





```
    while ( p != NULL )
    {
        z -> link = ( struct node * ) malloc (
                        sizeof ( struct node ) ) ;
        z = z -> link ;
        z -> data = p -> data ;
        p = p -> link ;
    }
    while ( q != NULL )
    {
        z -> link = ( struct node * ) malloc (
                        sizeof ( struct node ) ) ;
        z = z -> link ;
        z -> data = q -> data ;
        q = q -> link ;
    }
    z -> link = NULL ;
} // merge( )
```



Operations On Linked List - IV

Yashavant Kanetkar

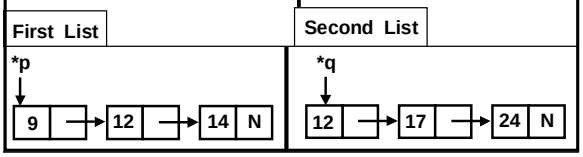
Objectives

- How to concatenate two linked lists
- How to perform the addition of polynomials by using linked lists

Concatenation Of LL

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct node  *f, *s ;
    f = s = NULL ;
    append ( &f, 1 ) ;
    append ( &f, 41 ) ;
    append ( &f, 3 ) ;
    append ( &f, 9 ) ;
    printf ( "\nFirst LL: " ) ;
    display ( f ) ;
    append ( &s, 7 ) ;
    append ( &s, 13 ) ;
    append ( &s, 2 ) ;
    append ( &s, 84 ) ;
    printf ( "\nSecond LL: " ) ;
    display ( second ) ;
    concat ( &f, &s ) ;
    printf ( "\nConcatenated LL: " ) ;
    display ( f ) ;
    deleteall ( f ) ;
    deleteall ( s ) ;
}
```

```
concat ( struct  node  **p,
           struct  node  **q )
{
    struct  node  *t ;
    if ( *p == NULL )
    {
        *p = *q ;
        return ;
    }
    if ( *q == NULL )
        return ;
    t = *p ;
    while ( t -> link != NULL )
        t = t -> link ;
    t -> link = *q ;
}
```



Polynomial Addition

```
#include "alloc.h"
struct  pnode
{
    float  coeff ;
    int  exp ;
    struct pnode  *link ;
} ;
main( )
{
    struct pnode  *f, *s, *t ;
    f = s = t = NULL ;
    pappend ( &f, 1.4, 5 ) ;
    pappend ( &f, 1.5, 4 ) ;
    pappend ( &f, 1.7, 2 ) ;
    pappend ( &f, 1.8, 1 ) ;
    pappend ( &f, 1.9, 0 ) ;
    pdisplay ( f ) ;
    pappend ( &s, 1.5, 6 ) ;
    pappend ( &s, 2.5, 5 ) ;
    pappend ( &s, -3.5, 4 ) ;
    pappend ( &s, 4.5, 3 ) ;
    pappend ( &s, 6.5, 1 ) ;
    pdisplay ( s ) ;
    padd ( f,  s,  &t ) ;
    pdisplay ( t ) ;
    deleteall ( f ) ;
    deleteall ( s ) ;
    deleteall ( t ) ;
}
```

```
padd ( struct pnode  *p, struct pnode  *q,   struct  pnode  **s )
{
    struct pnode  *z ;
    while ( p != NULL && q != NULL )
    {
        if ( *s == NULL )
            z = *s = ( struct pnode * ) malloc (
                                        sizeof ( struct pnode ) ) ;
        else
        {
            z -> link = ( struct pnode * ) malloc (
                                        sizeof ( struct pnode ) ) ;
            z = z -> link ;
        }
```

```
            if ( p -> exp == q -> exp )
            {
               z -> coeff = p -> coeff +  q -> coeff ;
               z -> exp = p -> exp ;
               p = p -> link ; q = q -> link ;
            }
            else
            {
               if ( p -> exp > q -> exp )
               {
                  *z = *p ;
                  p = p -> link ;
               }
               else
               {
                  *z = *q ;
                  q = q -> link ;
               }
            }
         }  // while
```

```
   while ( p != NULL )
   {
      z -> link = ( struct pnode * ) malloc (
                                    sizeof ( struct pnode ) ) ;
      z = z -> link ;
      *z = *p ;
      p = p -> link ;
   }
   while ( q != NULL )
   {
      z -> link = ( struct pnode * ) malloc (
                                    sizeof ( struct pnode ) ) ;
      z = z -> link ;
      *z = *q ;
      q = q -> link ;
   }
   z -> link = NULL ;
}  // padd
```

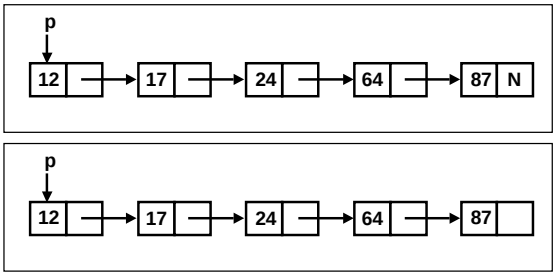
Circular Linked List

Yashavant Kanetkar

Objectives

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

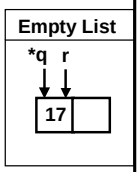
Circular Linked List

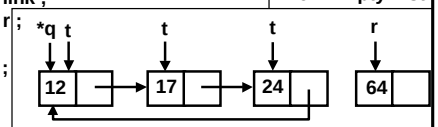


CLL Operations

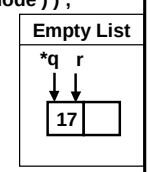
```
#include "alloc.h"
struct  node
{
     int  data ;
     struct  node *link ;
} ;
main( )
{
     struct  node  *p ;  int  c ;
     p = NULL ;
     cappend ( &p, 10 ) ;
     cappend ( &p, 18 ) ;
     caddatbeg ( &p, 5 ) ;
     caddatbeg ( &p, 15 ) ;
     caddafter ( &p, 2, 99 ) ;
     caddafter ( &p, 66, 88 ) ;
     cdisplay ( p ) ;
     cdel ( &p, 15 ) ;
     cdel ( &p, 10 ) ;
     cdel ( &p, 88 ) ;
     cdisplay ( p ) ;
     c = ccount ( p ) ;
     printf ( "\nNo. of ele. %d", c ) ;
     deleteall ( &p ) ;
}
```

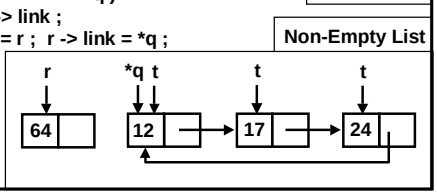
```
cappend ( struct  node  **q, int  n )
{
   struct  node  *r, *t ;
   r = ( struct node * ) malloc (
                  sizeof ( struct node ) ) ;
   r -> data = n ;
   if ( *q == NULL )
      *q = r ;
   else
   {
      t = *q ;
      while ( t -> link != *q )
         t = t -> link ;
      t -> link = r ;
   }
   r -> link = *q ;
}
```



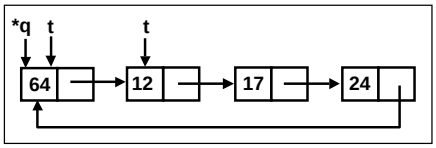



```
caddatbeg ( struct  node  **q, int  n )
{
     struct  node  *r, *t ;
     r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
     r -> data = n ;
     if ( *q == NULL )
          r -> link = r ;
     else
     {
          t = *q ;
          while ( t -> link != *q )
               t = t -> link ;
          t -> link = r ;  r -> link = *q ;
     }
     *q = r ;
}
```





```
caddafter ( struct  node  **q, int  pos, int  n )
{
    struct  node  *t ;
    int  i ;
    t = *q ;
    for ( i = 0 ; i < pos ; i++ )
    {
        if ( t -> link == *q )
            break ;
        t = t -> link ;
    }
```

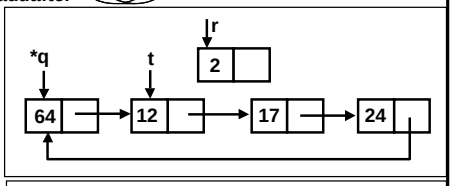


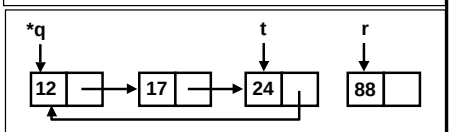
Cont…

Cont…

```
    r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    r -> data = n ;
    r -> link = t -> link ;
    t -> link = r ;
} // caddafter
```

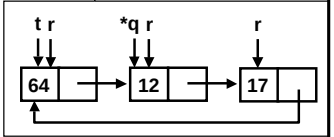
Define r






```
cdel ( struct  node  **q, int  n )
{
    struct  node  *t, *f ;
    t = *q ;   f = NULL ;
    while ( t != f )
    {
        f = *q ;
        if ( t -> data == n )
        {
            if ( t == *q )
            {
                r = *q ;
                while ( r -> link != *q )
                    r = r -> link ;
                *q = r -> link = t -> link ;
                free ( t ) ;
            }
```

Node to be Deleted = 64

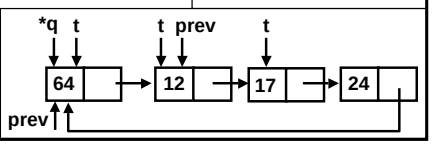


Cont…

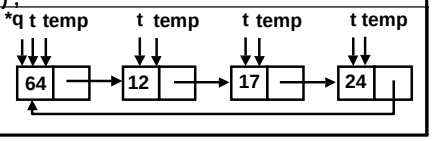
Cont...

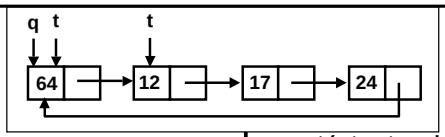
```
            else
            {
                prev -> link = t -> link ;
                free ( t ) ;
            }
            return ;
        } // if
        prev = t ;
        t = t -> link ;
    } // while
    printf ( "No. Not Found." ) ;
} // cdel
```

Node to be Deleted = 17




```
deleteall ( struct  node  **q )
{
  struct  node  *t, *last, *temp
  t = *q ;
  while ( t -> link != *q )
        t = t -> link ;
  last = t ;
  t = *q ;
  while ( t -> next != *q )
  {
     temp = t ;
     t = t -> link ;
     last -> link = *q = t
     free ( temp ) ;
  }
  free ( *q ) ;
  *q = NULL ;
}
```






```
cdisplay ( struct  node  *q )
{
    struct  node  *t , *f ;
    t = q ;
    f = NULL ;
     while ( t != f )
    {
        f = q ;
        printf ( "%d\t", t -> data ) ;
        t = t -> link ;
    }
}
```

```
ccount ( struct  node  *q )
{
    struct node  *t, *f ;
    int c = 0 ;
    t = q ;
    f = NULL ;
    while ( t != f )
    {
        f = q ; c++ ;
        t = t -> link ;
    }
    return  c ;
}
```

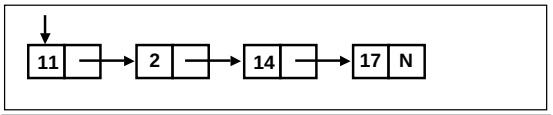
Doubly Linked List

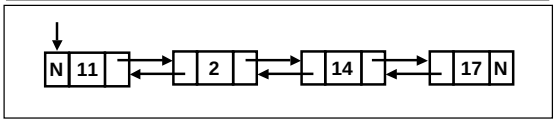
Yashavant Kanetkar

Objectives

- What are doubly linked lists
- How to perform similar common operations on a doubly linked list

Doubly Linked List



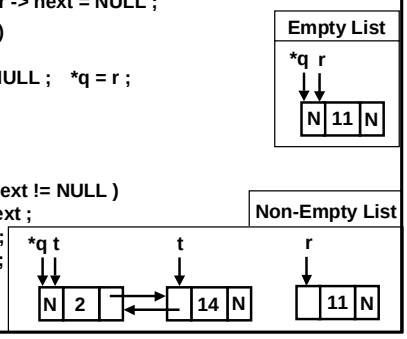



```
struct  dnode
{
    struct  dnode  *prev ;
    int  data ;
    struct  dnode  *next ;
}
```

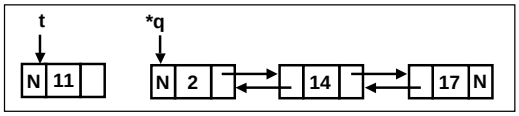
DLL Operations

```
#include "alloc.h"
struct  dnode
{
    struct  dnode  *prev ;
    int  data ;
    struct  dnode  *next ;
} ;
main( )
{
    struct  dnode  *p ; int  c ;
    p = NULL ;
    d_append ( &p , 11 ) ;
    d_append ( &p , 2 ) ;
    d_append ( &p , 14 ) ;
    d_append ( &p , 17 ) ;
    d_append ( &p , 99 ) ;
    d_display ( p ) ;
    d_addatbeg ( &p, 33 ) ;
    d_addatbeg ( &p, 55 ) ;
    d_display ( p ) ;
    d_addafter ( p, 4, 66 ) ;
    d_addafter ( p, 2, 96 ) ;
    d_display ( p ) ;
    d_delete ( &p, 55 ) ;
    d_delete ( &p, 2 ) ;
    d_delete ( &p, 99 ) ;
    d_display ( p ) ;
    c = d_count ( p ) ;
    printf ( "\nNo. of ele. %d", c ) ;
    deleteall ( &p ) ;
}
```

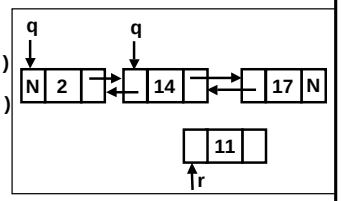
```
d_append ( struct  dnode  **q, int  n )
{
    struct  dnode  *r, *t ;
    r = ( struct  dnode * ) malloc ( sizeof ( struct  dnode ) ) ;
    r -> data = n ;   r -> next = NULL ;
    if ( *q == NULL )
    {
        r -> prev = NULL ;   *q = r ;
    }
    else
    {
        t = *q ;
        while ( t -> next != NULL )
            t = t -> next ;
        r -> prev = t ;
        t -> next = r ;
    }
}
```




```
d_addatbeg ( struct  dnode  **q, int  n )
{
    struct  dnode  *t ;
    t = ( struct  dnode * ) malloc ( sizeof ( struct  dnode ) ) ;
    t -> prev = NULL ;
    t -> data = n ;
    t -> next = *q ;
    ( *q ) -> prev = t ;
    *q = t ;
}
```



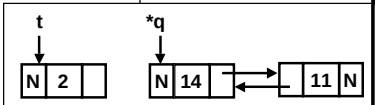
```
d_addafter ( struct  dnode  *q, int  pos, int  n )
{
    int i ;
    struct  dnode  *r ;
    for ( i = 0 ; i < pos ; i++ )
    {
       if ( q -> link == NULL )
            break ;
       q = q -> next ;
    }
   r = ( struct  dnode  * ) malloc (
                    sizeof ( struct  dnode ) ) ;
    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

```
d_delete ( struct  dnode  **q, int  n )
{
    struct  dnode  *t ;
    t = *q ;
    while ( t != NULL )
    {
        if ( t -> data == n )
        {
            if ( t == *q )
            {
                *q = ( *q ) -> next ;
                ( *q ) -> prev = NULL ;
            }
```

Node to be Deleted = 2

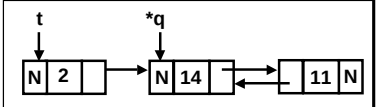


Cont...

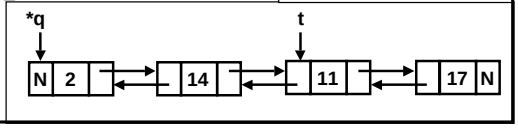
Cont...

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

Node to be Deleted = 2



Node to be Deleted = 11



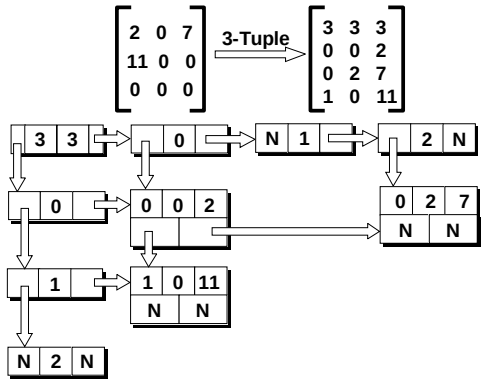
Sparse Matrices as Linked List I

Yashavant Kanetkar

Objectives

- How sparse matrices can be represented in the form of linked lists

SM Represented As LL



Exercise

$$\begin{bmatrix} 0 & 9 & 0 & 0 \\ 5 & 17 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{3-Tuple}} \begin{bmatrix} 3 & 4 & 3 \\ 0 & 1 & 9 \\ 1 & 0 & 5 \\ 1 & 1 & 17 \end{bmatrix}$$

Find the linked representation of given SM

Sparse Matrices as Linked List - II

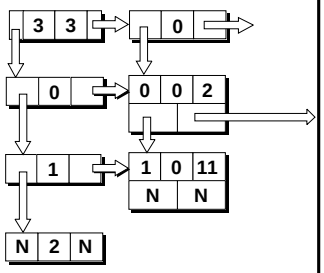
Yashavant Kanetkar

Objectives

- How to add a new node to the linked list
- How to display the contents of the linked list
- How to delete all the nodes in the linked list

Program

```
struct  node
{
    int  row, col, val ;
    struct  node *d, *r ;
} ;
struct  rcnode
{
    int  no ;
    struct  rcnode  *p1 ;
    struct  node  *p2 ;
} ;
struct  spmat
{
    struct  rcnode  *fr ;
    int  rows, cols ;
    struct  rcnode  *fc ;
} ;
```

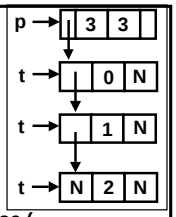


```
struct node
{
    int row, col, val ;
    struct node *d, *r ;
} ;
struct rcnode
{
    struct rcnode *p1 ;
    int no ;
    struct node *p2 ;
} ;
struct spmat
{
    struct rcnode *fr ;
    int rows, cols ;
    struct rcnode *fc ;
} ;
```

Program

```
#include "alloc.h"
main( )
{
    struct spmat s ; int i ;
    int tu [ 4 ] [ 3 ] = {
                            3, 3, 3,
                            0, 0, 2,
                            0, 2, 7,
                            1, 0, 11
                          } ;
    init ( &s, tu ) ;
    for ( i = 1 ; i <= tu[ 0 ][ 2 ] ; i++ )
        add ( s, tu + i ) ;
    display ( s ) ;
    deleteall ( &s ) ;
}
```

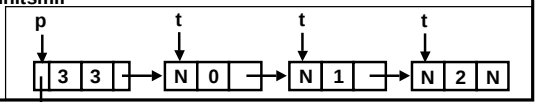
```
init ( struct spmat *p, int *ptup )
{
    struct rcnode *t ; int i ;
    p -> rows = ptup[ 0 ] ;
    p -> cols = ptup[ 1 ] ;
    for ( i = 0 ; i < ptup[ 0 ] ; i++ )
    {
        if ( i == 0 )
            t = p -> fr = ( struct rcnode * ) malloc (
                                    sizeof ( struct rcnode ) ) ;
        else
        {
            t -> p1 = ( struct rcnode * ) malloc ( sizeof (
                                        struct rcnode ) ) ;
            t = t -> p1 ;
        }
        t -> no = i ;  t -> p2 = NULL ;
    }
    t -> p1 = NULL ;
```



Cont...

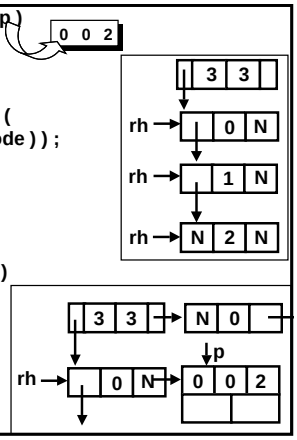
Cont...

```
    for ( i = 0 ; i < ptup[ 1 ] ; i++ )
    {
        if ( i == 0 )
            t = p -> fc = ( struct rcnode * ) malloc (
                                    sizeof ( struct rcnode ) ) ;
        else
        {
            t -> p1 = ( struct rcnode * ) malloc ( sizeof (
                                        struct rcnode ) ) ;
            t = t -> p1 ;
        }
        t -> no = i ;  t -> p2 = NULL ;
    }
    t -> p1 = NULL ;
} // initsmll
```

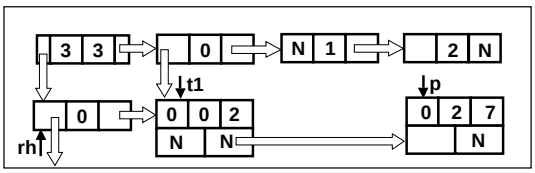


```
add ( struct  spmat  q, int  *ptup )
{
    struct  node  *p, *t1 ;
    struct  rcnode  *rh ;  int i ;

    p = ( struct  node * ) malloc (
                sizeof ( struct  node ) ) ;

    p -> row = ptup[ 0 ] ;
    p -> col = ptup[  1 ] ;
    p -> val = ptup[ 2 ] ;

    rh = q.fr ;
    for ( i = 0 ; i < p -> row ; i++ )
        rh = rh -> p1 ;
    t1 = rh -> p2 ;

    if ( t1 == NULL )
        rh -> p2 = p ;
```

Cont...



Cont...

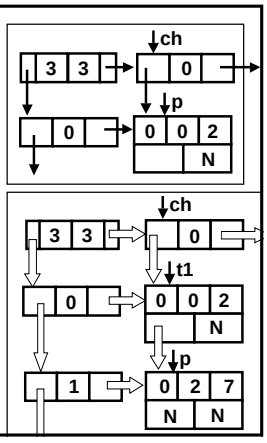



```
    else
    {
        while ( t1 -> r != NULL  )
            t1 = t1 -> r ;

        t1 -> r = p ;
    }
    p -> r = NULL ;
```

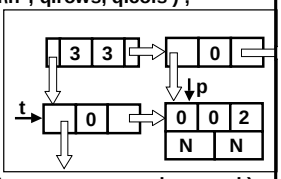
Cont...

Cont... Define ch

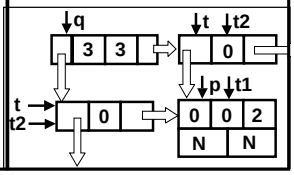
```
    ch = q.fc ;
    for ( i = 0 ; i < p -> col ; i++ )
        ch = ch -> p1 ;
    t1 = ch -> p2 ;

    if ( t1 == NULL )
        ch -> p2 = p ;
    else
    {
        while ( t1 -> d  !=  NULL )
            t1 = t1 -> d ;

        t1 -> d = p ;
    }
    p -> d = NULL ;
} // addspll
```



```
display ( struct  spmat  q )
{
    struct  rcnode  *t ;
    struct  node  *p ;

    printf ( "Row = %d, Col = %d\n", q.rows, q.cols ) ;

    t = q.fr ;

    while ( t != NULL )
    {
        p = t -> p2 ;
        while ( p != NULL )
        {
            printf ( "%d %d %d\n", p -> row, p -> col, p -> val ) ;
            p = p -> r ;
        }
        t = t -> p1 ;
    }
}
```




```
deleteall ( struct  spmat  *q )
{
    struct rcnode *t, *t2 ;
    struct node *p, *t1 ;

    t = q -> fr ;
    while ( t != NULL )
    {
        p = t -> p2 ;
        while ( p != NULL )
        {
            t1 = p ;
            p = p -> r ;
            free ( t1 ) ;
        }
        t2 = t ;
        t = t -> p1 ;
        free ( t2 ) ;
    }

    t = q -> fc ;
    while ( t != NULL )
    {
        t2 = t ;
        t = t -> p1 ;
        free ( t2 ) ;
    }
    q -> fr = q -> fc = NULL ;
}
```



Linked List Using Recursion

Yashavant Kanetkar

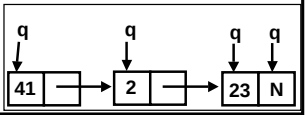
Objectives

- How to implement some of the linked list operations using recursion

Recursive *count( )*

```
#include "alloc.h"
struct  node
{
     int  data ;
     struct  node  *link ;
} ;
main( )
{
     struct  node  *p ;
     int  c ;
     p = NULL ;
     append ( &p, 41 ) ;
     append ( &p, 2 ) ;
     append ( &p, 23 ) ;
     c = count ( p ) ;
     printf ( "Ele. = %d", c ) ;
}
```

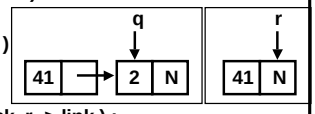
```
int count ( struct  node  *q )
{
     int  i ;
     if ( q != NULL )
     {
          i = 1 + count ( q -> link ) ;
          return ( i ) ;
     }
     else
          return ( 0 ) ;
}
```

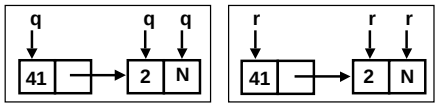


Recursive *compare( )*

```
# include  "alloc.h"
# define  SAME  1
# define  DIFF   0
struct  node
{
    int  data ;
    struct  node  *link ;
} ;

main( )
{
    struct  node  *f, *s ;
    f = s = NULL ;

    append ( &f, 1 ) ;
    append ( &f, 2 ) ;
    append ( &f, 3 ) ;

    append ( &s, 1 ) ;
    append ( &s, 2 ) ;
    append ( &s, 3 ) ;

    if ( compare ( f, s ) == SAME )
        printf ( "Equal." ) ;
    else
        printf ( "Not equal." ) ;

    deleteall ( &f ) ;
    deleteall ( &s ) ;
}
```

```
int  compare ( struct  node  *q, struct  node  *r )
{
  if ( ( q == NULL ) && ( r == NULL ) )
      return  SAME  ;
  else
  {
      if ( q == NULL || r == NULL )
          return  DIFF ;
      if ( q -> data != r -> data )
          return  DIFF ;
      else
          i = compare ( q -> link, r -> link ) ;
          return  i ;
  }
}
```

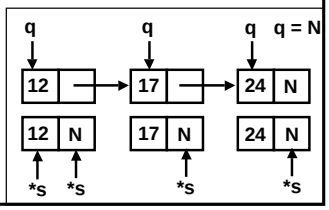




Recursive *copy( )*

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;

main( )
{
    struct  node  *f, *s ;
    f = s = NULL ;

    append ( &f, 17 ) ;
    append ( &f, 42 ) ;
    append ( &f, 3 ) ;
    append ( &f, 64 ) ;

    append ( &f, 51 ) ;
    append ( &f, 29 ) ;
    append ( &f, 7 ) ;

    display ( f ) ;
    copy ( f, &s ) ;

    display ( s ) ;
    deleteall ( &f ) ;
    deleteall ( &s ) ;
}
```

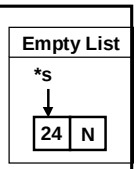
```
copy ( struct  node  *q, struct  node  **s )
{
    if ( q != NULL )
    {
        *s = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        ( *s ) -> data = q -> data ;
        ( *s ) -> link = NULL ;
        copy ( q -> link, &( ( *s ) -> link ) ) ;
    }
}
```

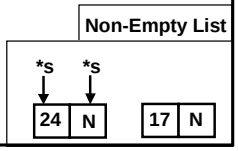


Recursive *append( )*

```
#include "alloc.h"
struct  node
{
    int  data ;
    struct  node  *link ;
} ;
main( )
{
    struct  node  *p ;
    p = NULL ;
    append ( &f, 17 ) ;
    append ( &f, 42 ) ;
    append ( &f, 3 ) ;
    append ( &f, 64 ) ;
    append ( &f, 51 ) ;
    append ( &f, 29 ) ;
    append ( &f, 7 ) ;

    display ( p ) ;
    deleteall ( &p ) ;
}
```

```
append ( struct  node  **s, int  n )
{
    if ( *s == NULL )
    {
        *s = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        ( *s ) -> data = n ;
        ( *s ) -> link = NULL ;
    }
    else
        append ( &( ( *s ) -> link ), n ) ;
}
```





Linked List Using Unions

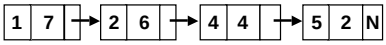
Yashavant Kanetkar

Objectives

- What are unions
- Declaring and using unions
- What are generalized linked lists

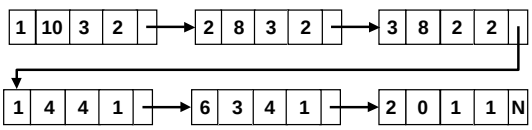
Problem

$x^7 + 2x^6 + 4x^4 + 5x^2$



$x^{10} y^3 z^2 + 2x^8 y^3 z^2 + 3x^8 y^2 z^2 + x^4 y^4 z + 6x^3 y^4 z + 2yz$

Solution I

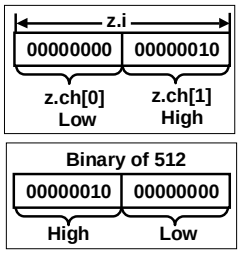


Each node contains a term

Unions

Permits access to same memory locations in more than one way

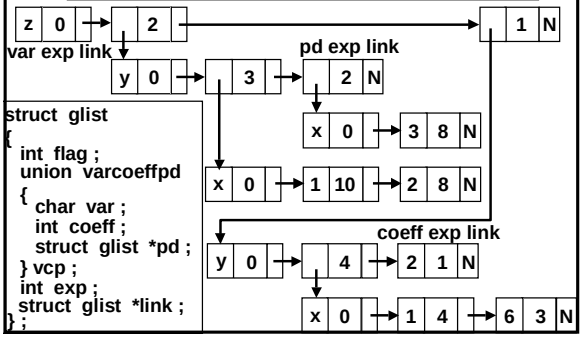
```
main( )
{
   union  a
   {
      int  i ;
      char  ch[ 2 ] ;
   } ;
   union  a  z ;
   z.i = 512 ;
   printf ( "%d", sizeof ( z ) ) ;
   printf ( "%d %d %d", z.i, z.ch[ 0 ], z.ch[ 1 ] ) ;
}
```



Solution II - Generalized List

$x^{10} y^3 z^2 + 2x^8 y^3 z^2 + 3x^8 y^2 z^2 + x^4 y^4 z + 6x^3 y^4 z + 2yz$

$((x^{10} + 2x^8) y^3 + 3x^8 y^2) z^2 + ((x^4 + 6x^3) y^4 + 2y) z$




```
struct  glist
{
  int  flag ;
  union  varcoeffpd
  {
    char  var ;
    int  coeff ;
    struct  glist  *pd ;
  } vcp ;
  int  exp ;
  struct  glist  *link ;
} ;
```

Exercise - I

$2a^4b^6c^3 + 5b^3c^3 + 9a^2b^2c - 3ab^2c + abc$

$(2a^4b^6 + 5b^3) c^3 + ((9a^2 - 3a) b^2 + (a) b) c$

Exercise - II

$3p^7q^9r^5 + 7p^5q^9r^5 - p^5q^8r^5 + 5p^2q^2r^2 + 2pq^2r^2 - 2qr$

$((3p^7 + 7p^5) q^9 + (- p^5) q^8) r^5 + ((5p^2 + 2p) q^2) r^2 + (- 2q) r$

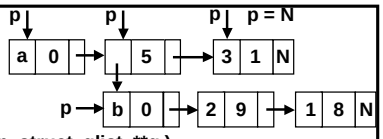
Generalized Linked List

Yashavant Kanetkar

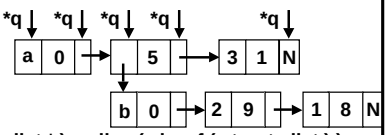
Objectives

- Copying and comparing generalized linked lists

Copy

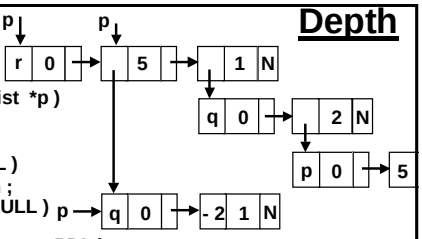



```
# define  VAR  0
# define  COF  1
# define  PDL  2
copy ( struct  glist  *p, struct  glist  **q )
{
    if ( p == NULL )
        *q = NULL ;
    else
    {
        *q = ( struct glist * ) malloc ( sizeof ( struct glist ) ) ;
        **q = *p ;
        if ( ( *q ) -> flag == PDL )
            copy ( p -> vcp.pd, &( ( *q ) -> vcp.pd ) ) ;
        copy ( p -> link, &( ( *q ) -> link ) ) ;
    }
}
```



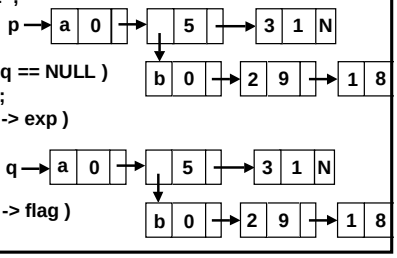
Depth

```
# define  VAR  0
# define  COF  1
# define  PDL  2
depth ( struct  glist  *p )
{
     int  m = 0, n ;
     if ( p == NULL )
          return  m ;
     while ( p != NULL )
     {
          if ( p -> flag == PDL )
          {
               n = depth ( p -> vcp.pd ) ;
               if ( n > m )
                    m = n ;
          }
          p = p -> link ;
     }
     return  ( m + 1 ) ;
}
```



Compare

```
# define  DIFF  0
# define  SAME  1

int  compare ( struct  glist  *p, struct   glist  *q )
{
     if ( ( p == NULL ) && ( q == NULL ) )
          return  SAME  ;

     if ( p == NULL || q == NULL )
          return  DIFF  ;
     if ( p -> exp != q -> exp )
          return  DIFF ;

     if ( p -> flag != q -> flag )
          return  DIFF ;
```



...Contd

```
          switch ( p -> flag )
          {
               case  VAR :
                    if ( p -> var != q -> var )
                         return  DIFF ;
                    break ;

               case  COF :
                    if ( p -> coeff != q -> coeff )
                         return  DIFF ;
                    break ;

               case  PDL :
                    i = compare (  p -> vcp.pd, q -> vcp.pd ) ;

                    if ( i == DIFF )
                         return  DIFF ;
          }
```

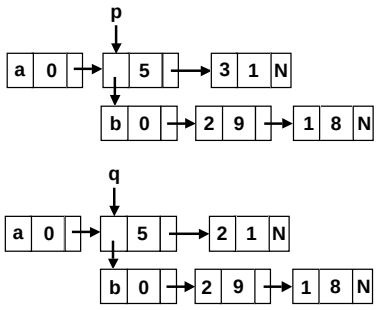
```
# define  VAR  0
# define  COF  1
# define  PDL  2
```

int i

Contd...

...Contd

```
        i = compare ( p -> link, q -> link ) ;
        return  i ;
}
```



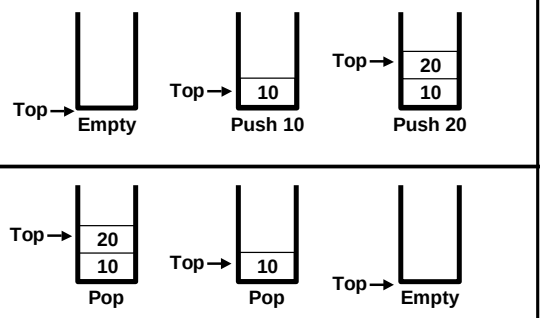
Stack

Yashavant Kanetkar

Objectives

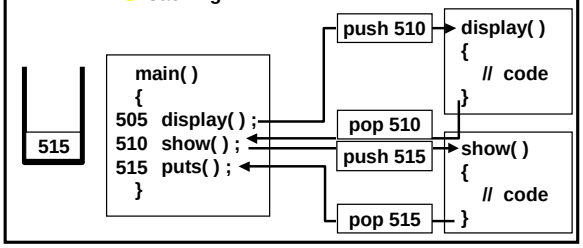
- Stack and its utilities
- Stack as an Array

Stack



Utility

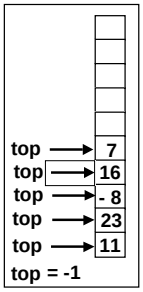
- Store local variables in a function
- Manage function calls
- Conversion of Expressions
- Evaluation of Expressions
- Caching




```
#define  NULL  0
#define  MAX  10
struct  stack
{
     int  arr[ MAX ] ;
     int  top ;
} ;
main( )
{
     struct  stack s ;  int  i, c ;
     s.top = -1 ;
     push ( &s, 11 ) ;
     push ( &s, 23 ) ;
     push ( &s, - 8 ) ;
     push ( &s, 16 ) ;
     push ( &s, 7 ) ;
     i = pop ( &s ) ;
     printf ( "%d", i ) ;
```

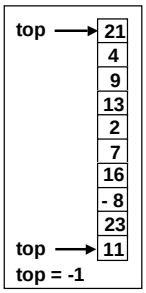
Program

```
     c = count ( s ) ;
     printf ( "Count = %d", c ) ;
     displaystack ( s ) ;
}
```



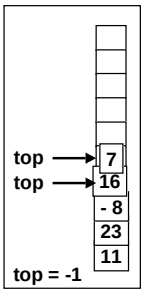
Push Operation

```
push ( struct  stack  *p, int  item )
{
     if ( p -> top == MAX - 1 )
     {
          printf ( "Stack is full." ) ;
          return ;
     }
     p -> top + + ;
     p -> arr[ p ->top ] = item ;
}
```



Pop Operation

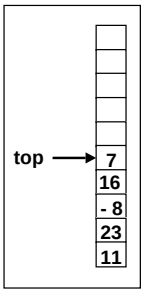
```
pop ( struct  stack  *p )
{
     int  data ;
     if ( p -> top == -1 )
     {
          printf ( "Stack is empty." ) ;
          return NULL ;
     }
     data = p -> arr[ p -> top ] ;
     p -> top - -  ;
     return data ;
}
```



Display And Count

```
displaystack ( struct  stack  q )
{
   int  i ;
   for ( i = 0 ; i <= q.top ; i + + )
      printf ( "%d\t", q.arr[ i ] ) ;
}
```

```
int count ( struct  stack  q )
{
   int  i = 0 ;
   while ( i <= q.top )
      i + + ;
   return  i ;
}
```



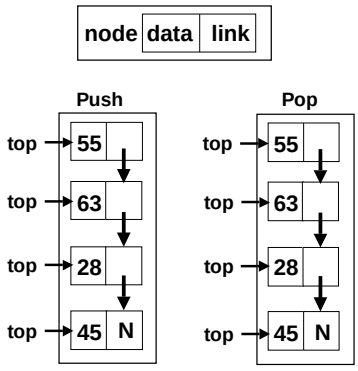
Stack As A Linked List

Yashavant Kanetkar

Objectives

- **Stack as a Linked List**

Stack As A Linked List

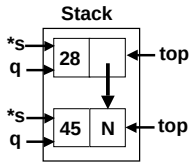


Program

```
#include "alloc.h"
struct  node
{
   int  data ;
   struct  node  *link ;
} ;

main( )
{
   struct  node  *top ;
   int  i, item ;
   top = NULL ;
   push ( &top, 45 ) ;
   push ( &top, 28 ) ;
   push ( &top, 63 ) ;
   push ( &top, 55 ) ;

   displaystack ( top ) ;
   t = count ( top ) ;
   printf ( "Total items = %d" , t ) ;

   printf ( "\nPopped : " ) ;
   item = pop ( &top ) ;
   printf ( "%d ", item ) ;

   displaystack ( top ) ;
   t = count ( top ) ;
   printf ( "Total items = %d" , t ) ;
}
```

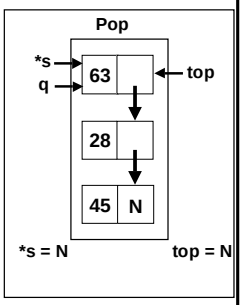
Push Operation

```
int push ( struct  node  **s, int  item )
{
   struct  node  *q ;
   q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
   q -> data = item ;
   q -> link = *s ;
   *s = q ;
}
```



Pop Operation

```
int pop ( struct  node  **s )
{
   struct  node  *q ;  int  item ;
   if ( *s == NULL )
   {
      printf ( "Stack is empty" ) ;
      return  NULL ;
   }
   q = *s ;
   item = q -> data ;
   *s = q -> link ;
   free ( q ) ;
   return ( item ) ;
}
```



Display And Count

```
void  displaystack ( struct  node  *q )
{
   while ( q != NULL )
   {
      printf ( "%2d ", q -> data ) ;
      q = q -> link ;
   }
}
```

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

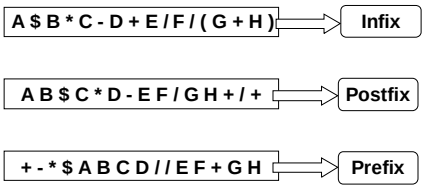
Stack Expressions

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 + / +

Stack Operations

Yashavant Kanetkar

Objectives

- **Infix to Postfix Conversion**

Program

```
#include  "string.h"
#include  "ctype.h"

#define  MAX  50

struct  post
{
      char  stack[ MAX ] ;
      int  top ;
      char  expr[ MAX ] ;
} ;

main( )
{
      struct  post  p ;
      char  infix[ MAX ] ;
      p.top = -1 ;
      printf ( "\nEnter Infix form: " ) ;
      gets ( infix ) ;
      convert ( infix, &p ) ;
      printf ( "Postfix expr. " ) ;
      printf ( "%s ", p.expr ) ;
}
```

Infix

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

Postfix

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

```
convert ( char  *in, struct  post  *q )
{
    char  *pt = q -> expr ;
    while ( *in != '\0' )
    {
        if ( *in == ' ' || *in == '\t' )
            in + + ;
        if ( isdigit ( *in ) || isalpha ( *in ) )
        {
            while ( isdigit ( *in ) || isalpha ( *in ) )
            {
                *pt = *in ;
                in + + ;  pt + + ;
            }
        }
```

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

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

Contd...

...Contd.

```
if ( *in == '(' )
{
    push ( q, *in ) ;
    in + + ;
}
if ( isoperator ( *in ) )
{
    if ( q -> top != -1 )
    {
        opr = pop ( q ) ;
        while ( priority ( opr ) >= priority ( *in ) )
        {
            *pt = opr ;  pt + + ;
            opr = pop ( q ) ;
        }
        push ( q, opr ) ;
    }
    push ( q, *in ) ;
    in + + ;
}
```

char opr

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

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

Contd...

...Contd.

```
if ( *in == ')' )
{
    opr = pop ( q ) ;
    while ( opr  !=  '(' )
    {
        *pt = opr ;  pt + + ;
        opr =  pop ( q ) ;
    }
    in + + ;
}
} // while
while ( q -> top != -1 )
{
    opr = pop ( q ) ;
    *pt = opr ;  pt + + ;
}
*pt = '\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

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

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

push( ) And *pop( )*

```
push ( struct  post  *q, char  c )
{
     if ( q -> top == MAX - 1 )
     {
          printf ( "Stk. full." ) ;
          return ;
     }
     q -> top++ ;
     q -> stack[q -> top] = c ;
}
```

```
char pop ( struct  post  *q )
{
     char  item ;
     if ( q -> top == -1 )
     {
          printf ( "Stk. empty" ) ;
          return  -1 ;
     }
     item = q -> stack[q -> top] ;
     q -> top-- ;
     return  item ;
}
```

Postfix

Yashavant Kanetkar

Objectives

- **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

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  "math.h"
# include  "ctype.h "

main( )
{
     char  expr[ MAX ] ;
     int  val ;

     printf ( "\nEnter postfix expr.: " ) ;
     gets ( expr ) ;
     val = calculate ( expr ) ;
     printf ( "\nValue = %d", val ) ;
}
```

```
#define  MAX  50
struct  postfix
{
    int  stack[ MAX ] ;
    int  top ;
} ;
calculate ( char  *s )
{
    struct  postfix  pf ;
    int  res, n1, n2, n3 ;
    pf.top = -1 ;
    while ( *s != '\0' )
    {
        if ( isdigit ( *s ) )
        {
            res = *s - '0' ;
            push ( &pf, res ) ;
        }
    else
    {
        n1 = pop ( &pf ) ;
        n2 = pop ( &pf ) ;
        switch ( *s )
        {
           case '+' :
               n3 = n2 + n1 ;
               break ;
           ..
           case '$' :
               n3 = pow ( n2 , n1 ) ;
               break ;
        }
        push ( &pf, n3 ) ;
    }
    s++ ;
  }
  res = pop (  &pf ) ;
  return res ;
}
```

Infix To Prefix

Yashavant Kanetkar

Objectives

- **Infix to Prefix Conversion**

In To Pre

Scan from R to L. 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
)	)	Empty
H	)	H
+	) +	H
G	) +	G H
(	Empty	+ G H
/	/	+ G H

Tok.	Stack	Expression
F	/	F + G H
/	/ /	F + G H
E	/ /	E F + G H
+	+	/ / E F + G H
D	+	D / / E F + G H
-	+ -	D / / E F + G H

...Contd.

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

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

Infix To Prefix

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

Tok.	Stack	Expression
H	Empty	H
*	*	H
)	*)	H
G	*)	G H
*	*) *	G H
)	*) *)	G H
F	*) *)	F G H
$	*) *) $	F G H
E	*) *) $	E F G H
/	*) *) /	$ E F G H

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

```
convert ( char  *in, struct  pre  *q )
{
     char  *pt ;
     int  l = strlen ( in ) ;
     in = in + ( l - 1 ) ;
     pt = q -> expr + ( l - 1 ) ;
     *( q -> expr + l ) = '\0' ;
     while ( l != 0 )
     {
          if ( *in == ' ' || *in == '\t' )
          {
               in - - ;
               l - - ;
          }
          while ( isdigit ( *in ) || isalpha ( *in ) )
          {
               *pt = *in ;
               in - - ;  l - - ;  pt - - ;
          }
```

Contd...

Infix

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

Prefix

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

...Contd.

```
if ( *in == ')' )
{
     push ( q, *in ) ;
     in - - ;  l - - ;
}
if ( isoperator ( *in ) )
{
     if ( q -> top != -1 )
     {
          opr = pop ( q ) ;
          while ( priority ( opr ) > priority ( *in ) )
          {
               *pt = opr ;  pt - - ;
               opr = pop ( q ) ;
          }
          push ( q, opr ) ;
     }
     push ( q, *in ) ;
     in - - ;  l - - ;
}
```

char opr

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

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

Contd...

...Contd.

```
if ( *in == '(' )
{
     opr = pop ( q ) ;
     while ( opr != ')' )
     {
          *pt = opr ; pt - - ;
          opr = pop ( q ) ;
     }
     in - - ;  l - - ;
}
} // while
while ( q -> top != -1 )
{
     opr = pop ( q ) ;
     *pt = opr ; pt - - ;
}
pt + + ;
strcpy ( q -> expr, pt ) ;
}
```

G	) +	G H
(	Empty	+ G H

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

Postfix To Prefix

Yashavant Kanetkar

Objectives

- Evaluation of Prefix form
- Postfix to Prefix Conversion

Evaluate Prefix

Scan from R to L. Repeat step 1- 2

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

Pop stack to obtain result

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

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

Post To Pre

Scan from L to R. Repeat step 1- 2

Token	Operation
operand	Push to stack
operator	Pop stack into s1 Pop stack into s2 Concatenate operator s2 s1 Push result on stack
Pop stack to obtain result	

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

Tok.	Stack
A	A
B	A B
$	$ A B
C	$ A B, C
*	* $ A B C
D	* $ A B C, D

Tok.	Stack
-	- * $ A B C D
E	- * $ 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
H	- * $ A B C D, / E F, G, H

...Contd.

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

Tok.	Stack
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

Post To Pre

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

Tok.	Stack
A	A
B	A, B
C	A, B, C
*	A, * B C
D	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
-	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

Program

```
#include  "string.h"
#define  MAX  50
struct  postfix
{
    char stack[ MAX ][ MAX ] ;
    int  top ;
} ;
main( )
{
     struct  postfix  q ;
     char  expr[ MAX ] ;
     q.top = -1 ;
     printf ( "Enter postfix expr: " ) ;
     gets ( expr ) ;
     convert ( expr, &q ) ;
     printf ( "\nPrefix expr: " ) ;
     printf ( "%s", &q.stack[ 0 ][ 0 ] ) ;
}
```

Postfix

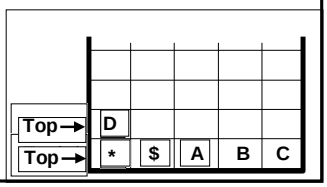
A B $ C * D - E F / G H + / +

Prefix

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

```
convert ( char  *str, struct  postfix  *p )
{
  char  temp[ MAX ] ;
  char  *s1, *s2 ;
  while ( *str != '\0' )
  {
     temp[ 0 ] = *str ;  temp[ 1 ] = '\0' ;
     if ( isoperator ( *str ) )
     {
        s1 = pop ( p ) ;
        s2 = pop ( p ) ;
        strcat ( temp, s2 ) ;
        strcat ( temp, s1 ) ;
     }
     push ( p, temp ) ;
     str + + ;
  }
}
```

A B $ C * D



push( )

```
push ( struct  postfix  *p, char  *str )
{
     if ( p -> top == MAX - 1 )
     {
          printf ( "\nStack is full." ) ;
          return ;
     }
     p -> top + + ;
     strcpy ( p -> stack[ p -> top ], str ) ;
}
```

pop( )

```
char * pop ( struct  postfix  *p )
{
     char  *pstr ;
     if ( p -> top == -1 )
     {
          printf ( "\nStack  is empty." ) ;
          return  0 ;
     }
     pstr = p -> stack[ p -> top ] ;
     p -> top - - ;
     return  pstr ;
}
```

Postfix To Infix

Yashavant Kanetkar

Objectives

➔Postfix to Infix Conversion

Post To In

Scan from L to R. Repeat step 1- 2	
Token	Operation
operand	Push to stack
operator	Pop stack into s1 Pop stack into s2 Concatenate s2 operator s1 Push result on stack
Pop stack to obtain result	

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

Tok.	Stack
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
F	A $ B * C - D, E, F
/	A $ B * C - D, E / F
G	A $ B * C - D, E / F, G
H	A $ B * C - D, E / F, G, H

...Contd.

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

Tok.	Stack
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

Post To In

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

Tok.	Stack
A	A
B	A, B
C	A, B, C
*	A, B * C
D	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
-	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

```
convert ( char  *str, struct  postfix  *p )
{
    char  temp[ MAX ] ;
    char  *s1, *s2 ;
    while ( *str != '\0' )
    {
        temp[ 0 ] = *str ;  temp[ 1 ] = '\0' ;
        if ( isoperator ( *str ) )
        {
            s1 = pop ( p ) ;
            s2 = pop ( p ) ;
            strcat ( s2,  temp ) ;
            strcat ( s2, s1 ) ;
            strcpy ( temp, s2 ) ;
        }
        push ( p, temp ) ;
        str + + ;
    }
}
```

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

Postfix

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

Infix

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

More Conversion

Yashavant Kanetkar

Objectives

- Prefix to Postfix Conversion
- Prefix to Infix Conversion

Pre To Post

Scan from R to L. Repeat step 1- 2

Token	Operation
operand	Push to stack
operator	Pop stack into s1 Pop stack into s2 Concatenate s1 s2 operator Push result on stack
Pop stack to obtain result	

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

Tok.	Stack
H	H
G	H, G
+	G H +
F	G H +, F
E	G H +, F, E
/	G H +, E F /

Tok.	Stack
/	E F / G H + /
D	E F / G H + /, D
C	E F / G H + /, D, C
B	E F / G H + /, D, C, B
A	E F / G H + /, D, C, B, A
$	E F / G H + /, D, C, A B $

...Contd.

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

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

Pre To Post

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

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

```
convert ( char  *str, struct  postfix  *p )
{
    char  temp[ MAX ] ;
    int  l ;  char  *s1, *s2 ;
    l = strlen ( str ) ;
    str = str + ( l - 1 ) ;
    while ( l  !=  0 )
    {
        temp[ 0 ] = *str ;  temp[ 1 ] = '\0' ;
        if ( isoperator ( *str ) )
        {
            s1 = pop ( p ) ; s2 = pop ( p ) ;
            strcat ( s1, s2 ) ;  strcat ( s1, temp ) ;
            strcpy ( temp, s1 ) ;
        }
        push ( p, temp ) ;
        str - - ;  l - - ;
    }
}
```

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

Prefix

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

Postfix

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

Pre To In

Scan from R to L. Repeat step 1- 2

Token	Operation
operand	Push to stack
operator	Pop stack into s1 Pop stack into s2 Concatenate s1 operator s2 Push result on stack
Pop stack to obtain result	

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

Tok.	Stack
H	H
G	H G
+	G + H
F	G + H, F
E	G + H, F, E
/	G + H, E / F
/	E / F / G + H
D	E / F / G + H, D
C	E / F / G + H, D, C
B	E / F / G + H, D, C, B
A	E / F / G + H, D, C, B, A
$	E / F / G + H, D, C, A $ B

...Contd.

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

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

Pre To In

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

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

```
convert ( char  *str, struct  infix  *p )
{
    char  temp[ MAX ] ;
    int  l ;  char  *s1, *s2 ;
    l = strlen ( str ) ;
    str = str + ( l - 1 ) ;
    while ( l != 0 )
    {
        temp[ 0 ] = *str ;  temp[ 1 ] = '\0' ;
        if ( isoperator ( *str ) )
        {
            s1 = pop ( p ) ; s2 = pop ( p ) ;
            strcat ( s1, temp ) ;  strcat ( s1, s2 ) ;
            strcpy ( temp, s1 ) ;
        }
        push ( p, temp ) ;
        str - - ;  l - - ;
    }
}
```

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

Prefix

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

Infix

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

Summary

Conversion	Scan Order	Token	Operation
In - Post	L - R	operand	Add to expression
		operator	Priority
In - Pre	R - L	(	Add to stack / delete
		)	Add to stack / delete
Post - Pre	L - R	operand	Push to stack
Post - In	L - R	operator	Pop stack into s1
Pre - Post	R - L		Pop stack into s2
Pre - In	R - L		Construct expression
			Push expression on stack

In
A $ B

Post
A B $

s1 → B
s2 → A

Pre
$ A B

s1 → A
s2 → B

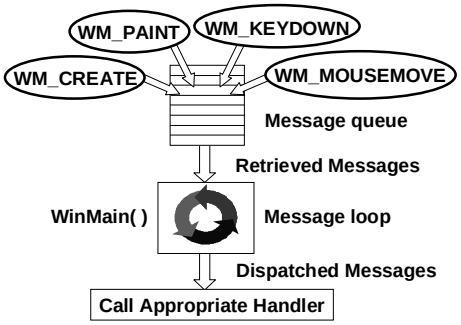
Queue - I

Yashavant Kanetkar

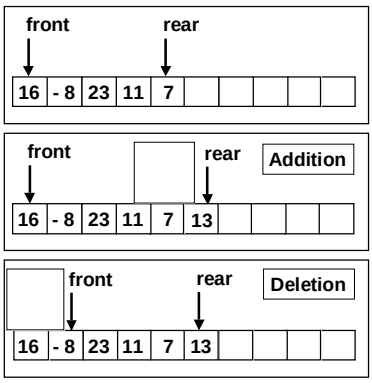
Objectives

- **Queue**

Queue



Queue

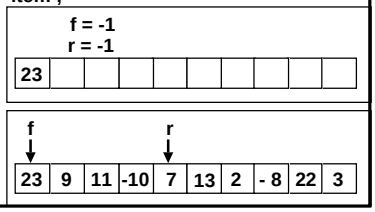


Program

```
#define  MAX  10
#define  NULL  0
struct  queue
{
     int  arr[ MAX ] ;
     int  f, r ;
} ;
main( )
{
     struct  queue  q ;  int  i ;
     q.f = q.r = -1 ;
     addq ( &q, 23 ) ;
     addq ( &q, 9 ) ;
     addq ( &q, 11 ) ;
     addq ( &q, -10 ) ;
     i = delq ( &q ) ;
     printf ( "\nItem = %d", i ) ;
}
```

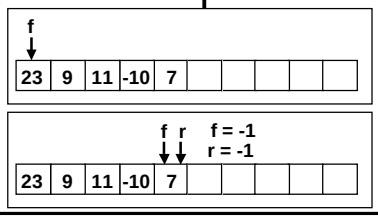
addq( )

```
addq ( struct  queue  *p, int  item )
{
     if ( p -> r == MAX - 1 )
     {
          printf ( "\nQueue is full." ) ;
          return ;
     }
     p -> r++ ;
     p -> arr[ p -> r ] = item ;
     if ( p -> f == -1 )
          p -> f = 0 ;
}
```



delq( )

```
delq ( struct  queue  *p )
{
     int  data ;
     if ( p -> f == -1 )
     {
          printf ( "\nEmpty." ) ;
          return NULL ;
     }
     data = p -> arr[ p -> f ] ;
     if ( p -> f == p -> r )
          p -> f =  p -> r = -1 ;
     else
          p -> f++ ;
     return  data ;
}
```



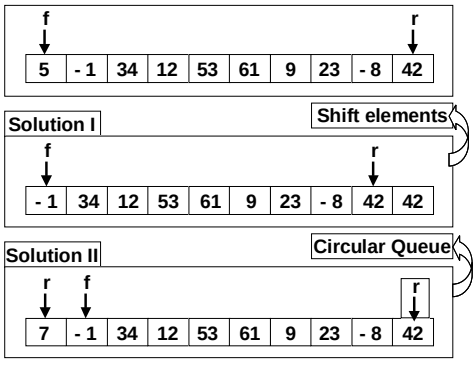
Queue - II

Yashavant Kanetkar

Objectives

- Limitations of Queue
- Circular Queues

Problems

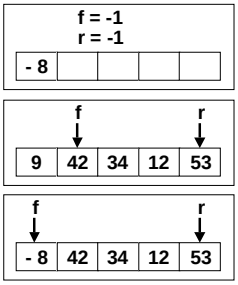


Program

```
#define  MAX  10
struct  queue
{
   int  arr[ MAX ] ;
   int  f, r ;
} ;
main( )
{
   struct  queue  q ;
   int  i ;

   q.f = q.r = -1 ;
   for ( i = 0 ; i < MAX ; i + + )
      q.arr[ i ] = 0 ;

   addq ( &q, 5 ) ;
   addq ( &q, - 1 ) ;
   addq ( &q, 34 ) ;
   addq ( &q, 12 ) ;
   addq ( &q, 53 ) ;

   printf ( "\nCircular queue: " ) ;
   for ( i = 0 ; i < MAX ; i + + )
      printf ( "%d\t", q.arr[ i ] ) ;

   i = delq ( &q ) ;
   printf ( "Item = %d", i ) ;

   printf ( "\nCircular queue: " ) ;
   for ( i = 0 ; i < MAX ; i + + )
      printf ( "%d\t", q.arr[ i ] ) ;
}
```

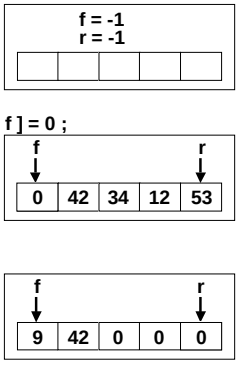
addq( )

```
addq ( struct  queue  *p, int  item )
{
   if ( ( p -> r == MAX - 1 && p -> f == 0 ) ||
        ( p -> r + 1 == p -> f ) )
   {
      printf ( "\nQueue is full." ) ;
      return ;
   }
   if ( p -> r == MAX - 1 )
      p -> r = 0 ;
   else
      p -> r + + ;
   p -> arr[ p -> r ] = item ;
   if ( p -> f == -1 )
      p -> f = 0 ;
}
```



delq( )

```
delq ( struct  queue  *p )
{
   int  i ;
   if ( p -> f == -1 )
   {
      printf ( "\nQueue is empty." ) ;
      return  0 ;
   }
   i = p -> arr[ p -> f ] ; p -> arr[ p -> f ] = 0 ;
   if ( p -> f == p -> r )
      p -> r = p -> f = -1 ;
   else
   {
      if ( p -> f == MAX - 1 )
         p -> f = 0 ;
      else
         p -> f + + ;
   }
   return  i ;
}
```



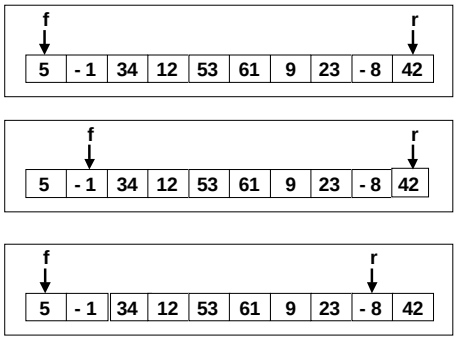
Deque & Priority Queue

Yashavant Kanetkar

Objectives

- Deque
- Priority Queue

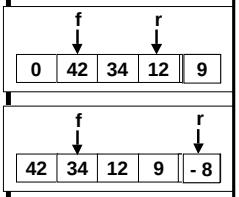
Deque



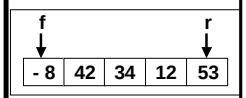
Program

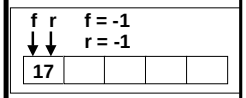
```
#define MAX 10
struct deque
{
   int arr[ MAX ] ;
   int f, r ;
} ;
main( )
{
   struct deque q ; int i ;
   dq.f = dq.r = -1 ;
   for ( i = 0 ; i < MAX ; i + + )
      dq.arr[ i ] = 0 ;
   addqatend ( &dq, 17 ) ;
   addqatbeg ( &dq, 10 ) ;
   addqatend ( &dq, 8 ) ;
   addqatbeg ( &dq, -9 ) ;
   addqatend ( &dq, 13 ) ;
   printf ( "\nDeque: " ) ;
   for ( i = 0 ; i < MAX ; i + + )
      printf ( "\t%d", dq.arr[ i ] ) ;

   i = delqatbeg ( &dq ) ;
   printf ( "\nItem = %d", i ) ;

   printf ( "\nDeque: " ) ;
   for ( i = 0 ; i < MAX ; i + + )
      printf ( "\t%d", dq.arr[ i ] ) ;

   i = delqatend ( &dq ) ;
   printf ( "\nItem = %d", i ) ;

   printf ( "\nDeque: " ) ;
   for ( i = 0 ; i < MAX ; i + + )
      printf ( "\t%d", dq.arr[ i ] ) ;
}
```

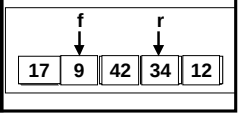
```
addqatend ( struct deque *p,
                          int item )
{
   int i ;
   if ( p -> f == 0 && p -> r == MAX-1 )
   {
      printf ( "Deque is full." ) ;
      return ;
   }
   if ( p -> f == -1 )
   {
      p -> r = p -> f = 0 ;
      p -> arr[ p -> r ] = item ;
      return ;
   }
   if ( p -> r == MAX - 1 )
   {
      for ( i = p -> f-1 ; i < p -> r ; i + + )
            p -> arr[ i ] = p -> arr[ i + 1 ] ;
      p -> f - - ;
   }
   else
      p -> r + + ;
   p -> arr[ p->r ] = item ;
}
```



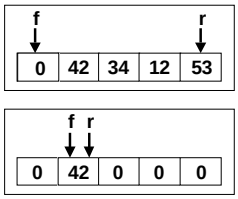

```
addqatbeg ( struct deque *p,
                          int item )
{
   int i ;
   if ( p -> f == 0 && p -> r == MAX-1 )
   {
      printf ( "Deque is full." ) ;
      return ;
   }
   if ( p -> f == -1 )
   {
      p -> f = p -> r = 0 ;
      p -> arr[ p -> f ] = item ;
      return ;
   }
   if ( p -> f == 0 )
   {
      for ( i = p -> r+1 ; i > p -> f ; i- - )
         p -> arr[ i ] = p -> arr[ i - 1 ] ;
      p -> r + + ;
   }
   else
      p -> f - - ;
   p -> arr[ p->f ] = item ;
}
```



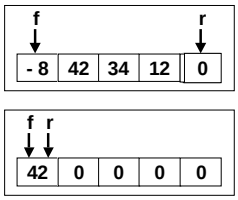




```
delqatbeg ( struct  deque  *p )
{
   int  item ;
   if ( p -> f == -1 )
   {
      printf ( "Deque is empty." ) ;
      return  0 ;
   }
   item = p -> arr[ p -> f ] ;
   p -> arr[ p -> f ] = 0 ;
   if ( p -> f == p -> r )
      p -> f = p -> r = -1 ;
   else
      p -> f + + ;
   return  item ;
}
```

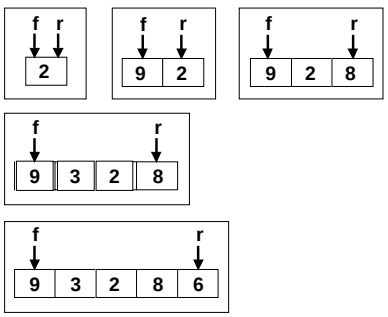



```
delqatend ( struct  deque  *p )
{
   int  item ;
   if ( p -> f == -1 )
   {
      printf ( "Deque is empty." ) ;
      return  0 ;
   }
   item = p -> arr[ p -> r ] ;
   p -> arr[ p -> r ] = 0 ;
   p -> r - - ;
   if ( p -> r == -1 )
      p -> f = -1 ;
   return  item ;
}
```



Priority Queue

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



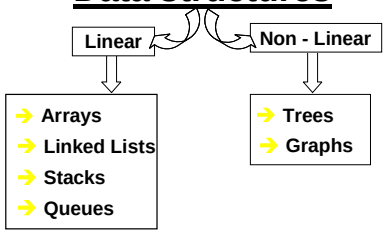
Trees

Yashavant Kanetkar

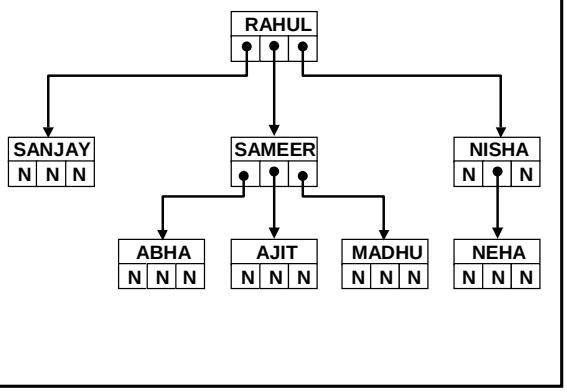
Objectives

- Trees
- Trees terminology
- Types of Trees

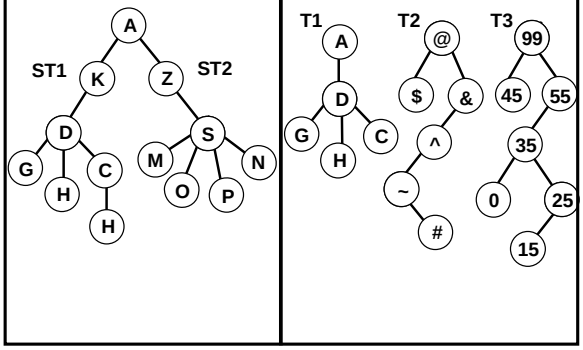
Data structures



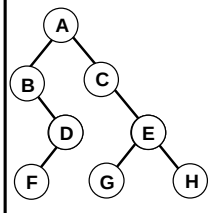
Trees



General Trees And Forest

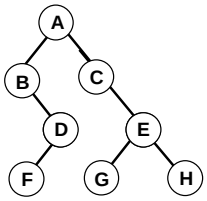


Terminology



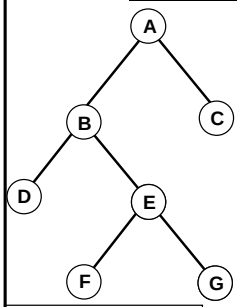
- Binary Tree - Each node has 0, 1, 2 children
- A - Root node
- B, D, F - Nodes of left sub-tree
- C, E, G, H - Nodes of right sub-tree
- F, G, H - Leaf nodes
- A is Father of B & C
- E is right child C
- G is right descendant of A
- F is left child of D
- F is left descendant of A
- A is ancestor of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

More Terminology



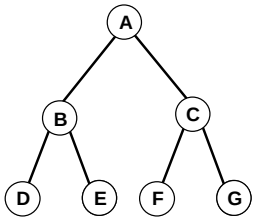
- Degree of a node is no. of nodes connected to it
- Level of root node is 0
- Level of any other node is 1 more than level of its father
- Depth of a binary tree is maximum level of a node

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

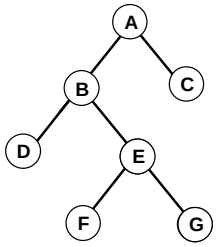
Tree Traversal

Yashavant Kanetkar

Objectives

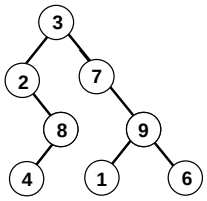
- Traversal of Trees
- Reconstruction of Trees

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

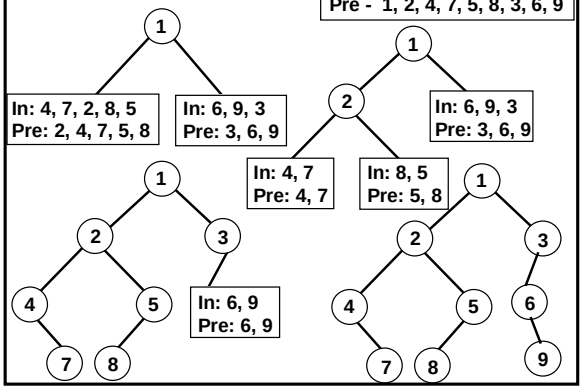
Exercise



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

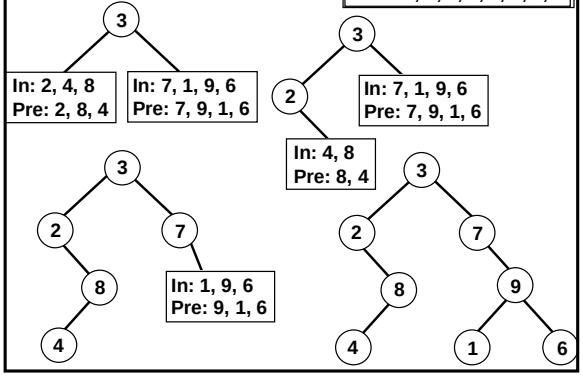
Reconstruction - I

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



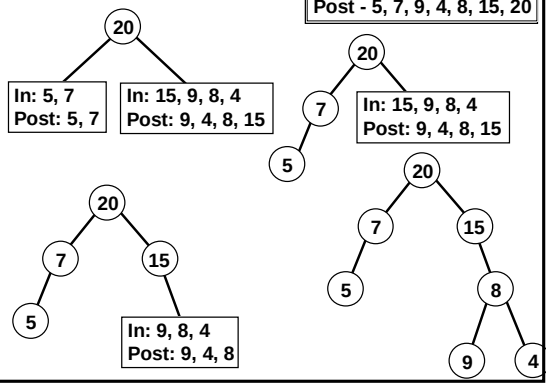
Exercise

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



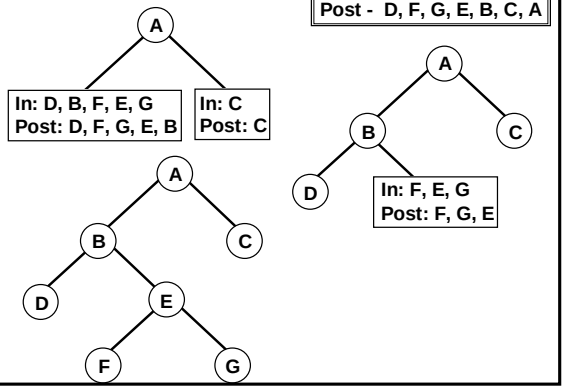
Reconstruction - II

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



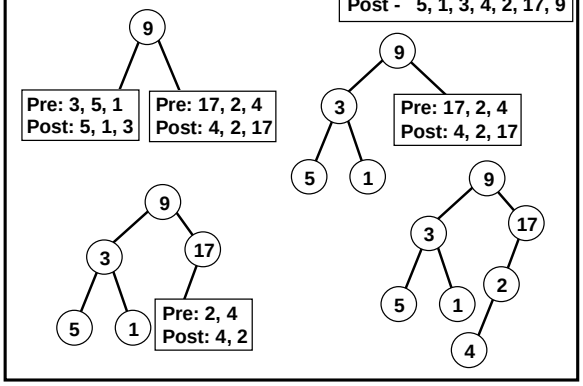
Exercise

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



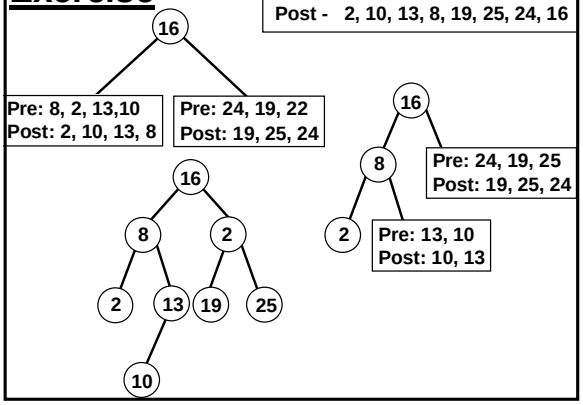
Reconstruction - III

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



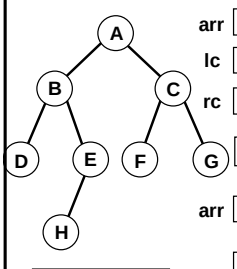
BST - I

Yashavant Kanetkar

Objectives

- How to represent trees using arrays and linked lists
- What are binary search trees

Array Representation



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

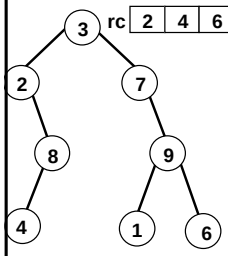
10	11	12	13	14	15	16	17	18	19
'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

20	21	22	23	24	25	26	27	28	29	30
'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'

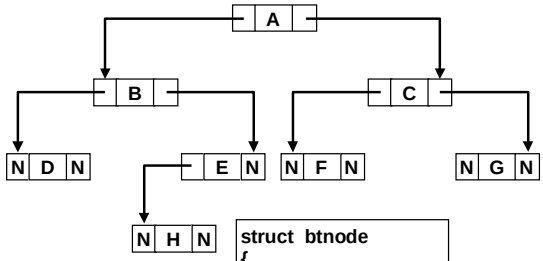
lc = 2 pos + 1
rc = 2 pos + 2

Exercise

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr	3	2	7	- 1	8	- 1	9	- 1	- 1	4	- 1	1	6
lc	1	- 1	- 1	- 1	9	- 1	11	- 1	- 1	- 1	- 1	- 1	- 1
rc	2	4	6	- 1	- 1	- 1	12	- 1	- 1	- 1	- 1	- 1	- 1

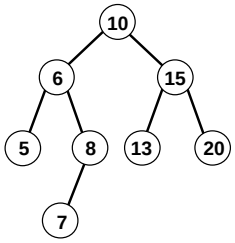


Linked Representation




```
struct  btnode
{
    struct  btnode  *lc ;
    char  data ;
    struct  btnode  *rc ;
} ;
```

Binary Search Tree



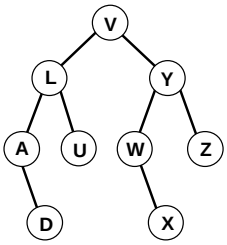
10, 6, 8, 15, 20, 7, 13, 5

Greater goes to right
Smaller goes to left

In-order gives ascending order

5, 6, 7, 8, 10, 13, 15, 20

Exercise



BST - II

Yashavant Kanetkar

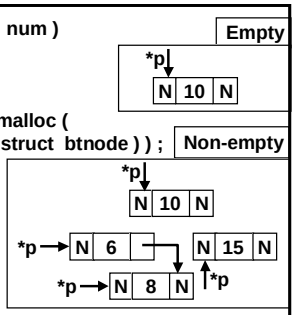
Objectives

- A program to traverse the trees
- Traversal of trees using inorder, preorder and postorder
- Non recursive traversal of trees
- Comparing two trees

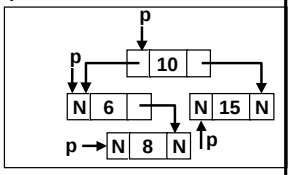
Program

```
# include  "alloc.h"
struct  btnode
{
    struct  btnode  *lc ;
    int  data ;
    struct  btnode  *rc ;
} ;
main( )
{
    struct  btnode  *bt ;

    bt = NULL ;

    insert ( &bt, 10 ) ;
    insert ( &bt, 6 ) ;
    insert ( &bt, 8 ) ;
    insert ( &bt, 15 ) ;
    insert ( &bt, 20 ) ;

    printf ( "\nIn-order: " ) ;
    inorder ( bt ) ;

    printf ( "\nPre-order: " ) ;
    preorder ( bt ) ;

    printf ( "\nPost-order: " ) ;
    postorder ( bt ) ;

    deleteall ( bt ) ;
}
```

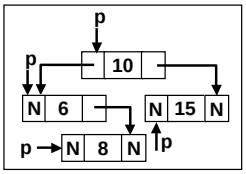
```
insert ( struct  btnode  **p, int  num )
{
    if ( *p == NULL )
    {
        *p = ( struct  btnode * ) malloc (
                        sizeof ( struct  btnode ) ) ;
        ( *p ) -> lc = NULL ;
        ( *p ) -> data = num ;
        ( *p ) -> rc = NULL ;
    }
    else
    {
        if ( num < ( *p ) -> data )
            insert ( &( ( *p ) -> lc ), num ) ;
        else
            insert ( &( ( *p ) -> rc ), num ) ;
    }
}
```




```
inorder ( struct  btnode  *p )
{
    if ( p != NULL )
    {
        inorder ( p -> lc ) ;
        printf ( "\t %d", p -> data ) ;
        inorder ( p -> rc ) ;
    }
}
preorder ( struct  btnode  *p )
{
    if ( p != NULL )
    {
        printf ( "\t %d", p -> data ) ;
        preorder ( p -> lc ) ;
        preorder ( p -> rc ) ;
    }
}
```



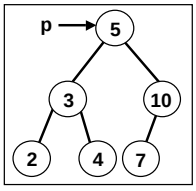

```
postorder ( struct  btnode  *p )
{
    if ( p != NULL )
    {
        postorder ( p -> lc ) ;
        postorder ( p -> rc ) ;
        printf ( "\t%d", p -> data ) ;
    }
}
deleteall ( struct  btnode  *p )
{
    if ( p != NULL )
    {
        deleteall ( p -> lc ) ;
        deleteall ( p -> rc ) ;
        free ( p ) ;
    }
}
```



Compare Trees

```
#include  "alloc.h "
# define TRUE  1
# define FALSE 0
struct  btnode
{
    struct  btnode  *lc ;
    int  data ;
    struct  btnode  *rc ;
} ;
main( )
{
    struct  btnode  *bt1, *bt2 ;
    int  i ;
    bt1 = bt2 = NULL ;
    insert ( &bt1, 5 ) ;
    insert ( &bt1, 3 ) ;
    insert ( &bt1, 10 ) ;
    insert ( &bt1, 4 ) ;
    insert ( &bt1, 2 ) ;

    insert ( &bt2, 5 ) ;
    insert ( &bt2, 3 ) ;
    insert ( &bt2, 10 ) ;
    insert ( &bt2, 4 ) ;
    insert ( &bt2, 2 ) ;

    compare ( bt1, bt2, &i ) ;
    if ( i == TRUE )
        printf ( "Equal" ) ;
    else
        printf ( "Unequal" ) ;
}
```

```
compare ( struct  btnode *p, struct  btnode  *q, int  *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 ) ;
        }
    }
}
```



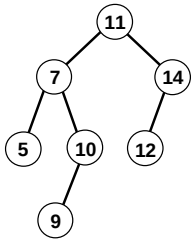
Binary Tree

Yashavant Kanetkar

Objectives

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

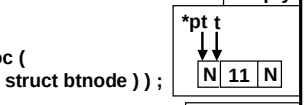
In - order successor of a node with two child is always a leaf node or a node with only right child

Program

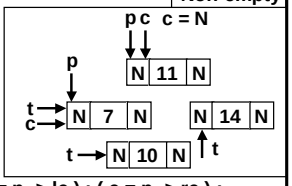
```
# include  "alloc.h"
struct  btnode
{
    struct  btnode  *lc ;
    int  data ;
    struct  btnode  *rc ;
} ;
main( )
{
    struct  btnode  *bt ;

    bt = NULL ;

    insert ( &bt, 11 ) ;
    insert ( &bt, 7 ) ;
    insert ( &bt, 10 ) ;
    insert ( &bt, 14 ) ;
    insert ( &bt, 5 ) ;
    insert ( &bt, 9 ) ;
    insert ( &bt, 12 ) ;

    printf ( "\nBST: " ) ;
    inorder ( bt ) ;

    delete ( &bt, 14 ) ;
    delete ( &bt, 7 ) ;
    delete ( &bt, 5 ) ;

    printf ( "\nBST: " ) ;
    inorder ( bt ) ;

    deleteall ( bt ) ;
}
```

```
insert ( struct  btnode  **pt, int  num )
{
    struct  btnode  *t, *p, *c ;

    t = ( struct btnode * ) malloc (
                        sizeof ( struct 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 ) ;
    }
}
```

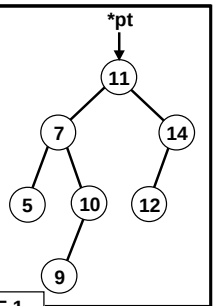
Empty



Non-empty




```
delete ( struct  btnode  **pt, int  num )
{
    struct  btnode  *p, *c ;
    int  found ;

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

    if ( found == FALSE )
    {
        printf ( "\nData not found" ) ;
        return ;
    }
```

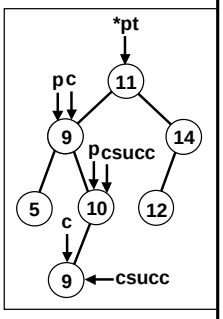



```
# define TRUE 1
# define FALSE 0
```

Contd...

...Contd

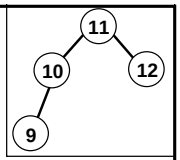
```
if ( c -> lc != NULL && c -> rc != NULL )
{
    p = c ;
    csucc = c -> rc ;
        struct  btnode  *csucc
    while ( csucc -> lc != NULL )
    {
        p = csucc ;
        csucc = csucc -> lc ;
    }
    c -> data = csucc -> data ;
    c = csucc ;
}
```

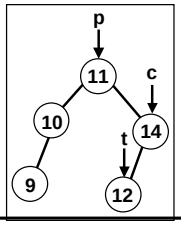


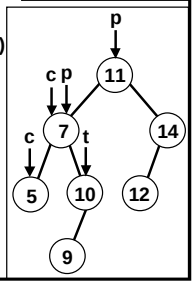
Contd...

...Contd

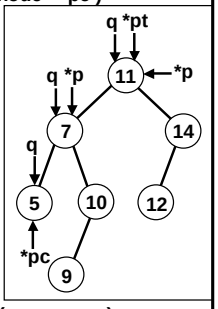
```
if ( c -> lc == NULL && c -> rc == NULL )
    t = NULL ;
        struct  btnode  *t
if ( c -> lc == NULL && c -> rc != NULL )
    t = c -> rc ;
if ( c -> lc != NULL && c -> rc == NULL )
    t = c -> lc ;
if ( p -> lc == c )
    p -> lc = t ;
else
    p -> rc = t ;
free ( c ) ;
}
```








```
search ( struct  btnode  **pt, int  num, struct  btnode  **p,
                                 struct  btnode  **pc )
{
    struct  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 ;
}
```



Treaded Binary Trees

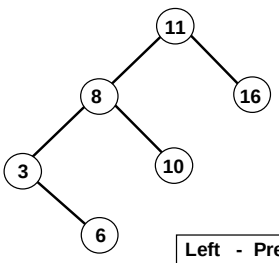
Yashavant Kanetkar

Objectives

- What are threaded binary trees
- How threaded binary trees are represented
- How to insert a node in a threaded binary tree
- How to traverse the threaded binary tree in inorder

Threaded Binary Tree

A. J. Perlis
C. Thornton

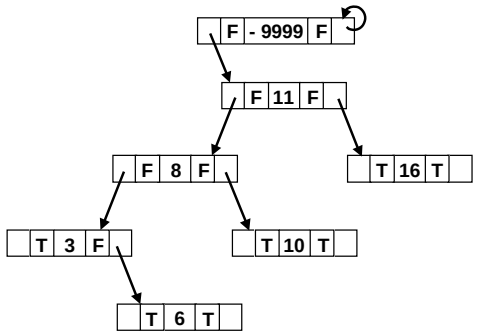


→ Links
→ Threads

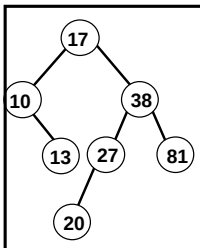
```
struct  thtree
{
    struct  thtree  *lc ;
    int  haslt ;
    int  data ;
    int  hasrt ;
    struct  thtree  *rc ;
} ;
```

Left - Predecessor or head
Right - Successor or head

Linked Representation



Exercise



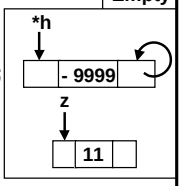
In: 10, 13, 17, 20, 27, 38, 81
Post: 13, 10, 20, 27, 81, 38, 17

Program

```
# include  "alloc.h"
# define   TRUE  1
# define  FALSE  0
 struct  thtree
{
     struct  thtree  *lc ;
     int  haslt ;
     int  data ;
     int  hasrt ;
     struct  thtree  *rc ;
} ;
main( )
{
     struct  thtree  *th_head ;
     th_head = NULL ;
     insert ( &th_head, 11 ) ;
     insert ( &th_head, 8 ) ;
     insert ( &th_head, 16 ) ;
     insert ( &th_head, 3 ) ;
     insert ( &th_head, 6 ) ;
     insert ( &th_head, 10 ) ;
     printf ( "Threaded BT:\n" ) ;
     inorder ( th_head ) ;
}
```

```
insert ( struct  thtree  **h, int  num )
{
    struct  thtree  *z ;
    z = ( struct  thtree * ) malloc (
                    sizeof ( struct  thtree ) ) ;
    z -> hasrt = z -> haslt = TRUE ;
    z -> data = num ;
    if ( *h == NULL )
    {
        *h = ( struct  thtree * ) malloc ( sizeof ( struct  thtree ) ) ;
        ( *h ) -> hasrt = ( *h ) -> haslt = FALSE ;
        ( *h ) -> lc = z ;
        ( *h ) -> data = - 9999 ;
        ( *h ) -> rc = *h ;
        z -> rc = z -> lc = *h ;

        return ;
    }
```

Empty

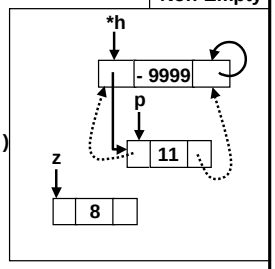


Contd...

...Contd

```
    p = ( *h ) -> lc ;
    struct  thtree  *p
    while ( p != *h )
    {
        if ( p -> data > num )
        {
            if ( p -> haslt == FALSE )
                p = p -> lc ;
            else
            {
                z -> lc = p -> lc ;
                z -> rc = p ;
                p -> lc = z ;
                p -> haslt = FALSE ;
                return ;
            }
        }
```

Non-Empty

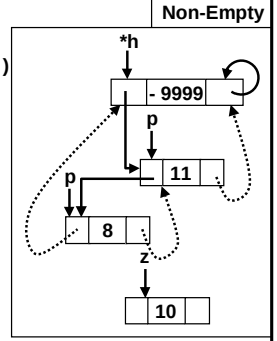


Contd...

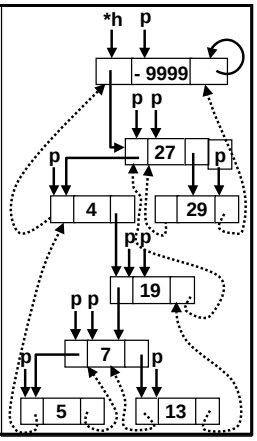
...Contd

```
        if ( p -> data < num )
        {
            if ( p -> hasrt == FALSE )
                p = p -> rc ;
            else
            {
                z -> rc = p -> rc ;
                z -> lc = p ;
                p -> rc = z ;
                p -> hasrt = FALSE ;
                return ;
            }
        }
    }
}
```

Non-Empty



```
inorder ( struct  thtree  *h )
{
    struct  thtree  *p ;
    p = h -> lc ;
    while ( p != h )
    {
        while ( p -> haslt == FALSE )
            p = p -> lc ;
        printf ( "%d ", p -> data ) ;
        while ( p -> hasrt == TRUE )
        {
            p = p -> rc ;
            if ( p == h )
                break ;
            printf ( "%d ", p -> data ) ;
        }
        p = p -> rc ;
    }
}
```



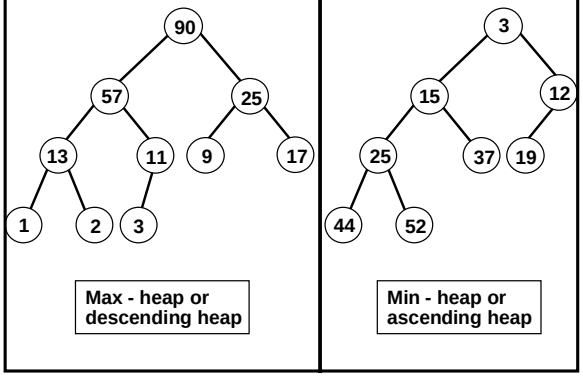
Heap

Yashavant Kanetkar

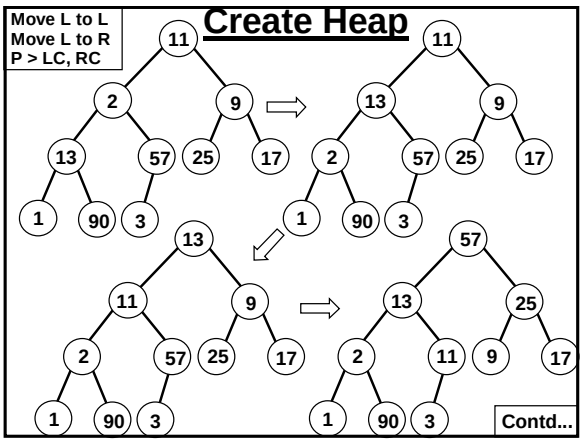
Objectives

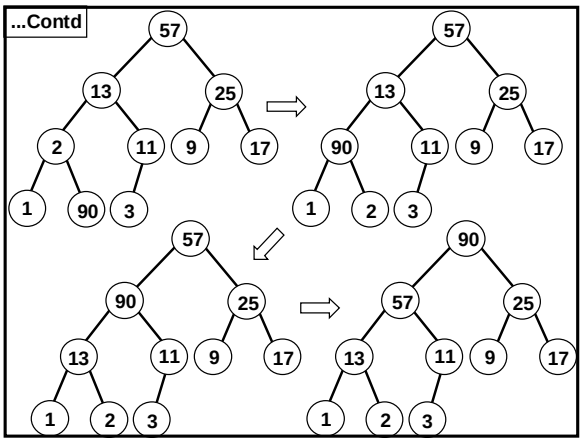
- What is a heap
- How to construct a heap
- How to perform the heap sort

Heap

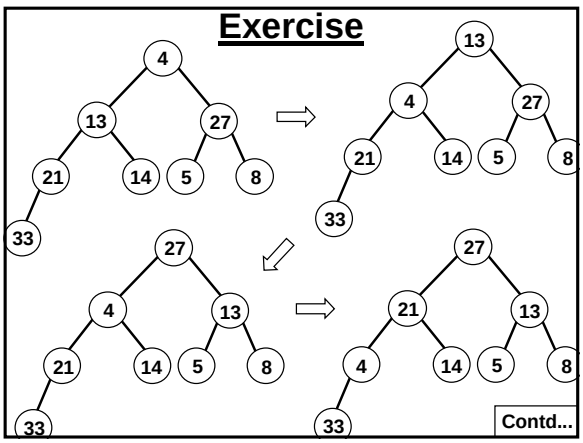


Create Heap



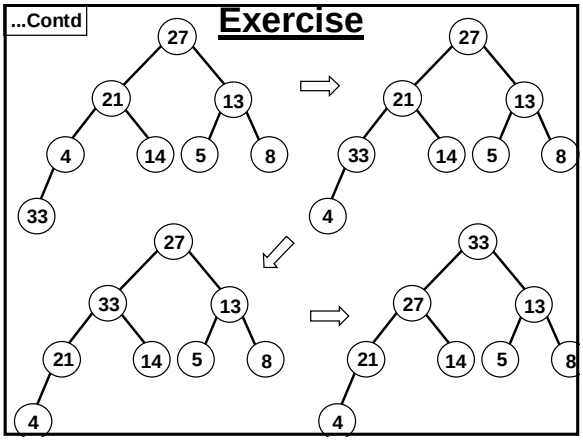


Exercise



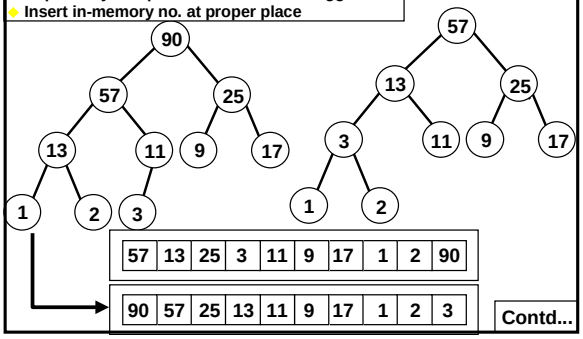
...Contd

Exercise



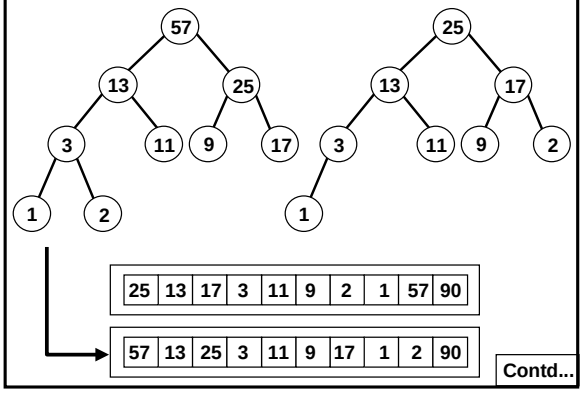
Heap Sort

- Memorize highest index value
- Replace it with root value
- Determine bigger child
- Shift it to root
- Repeatedly compare in-mem no. with bigger child
- Insert in-memory no. at proper place



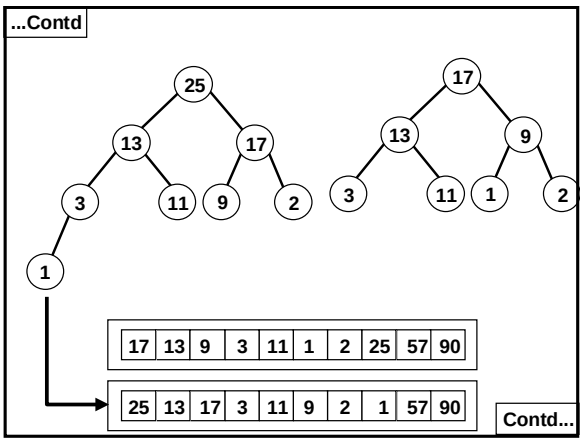
Contd...

...Contd

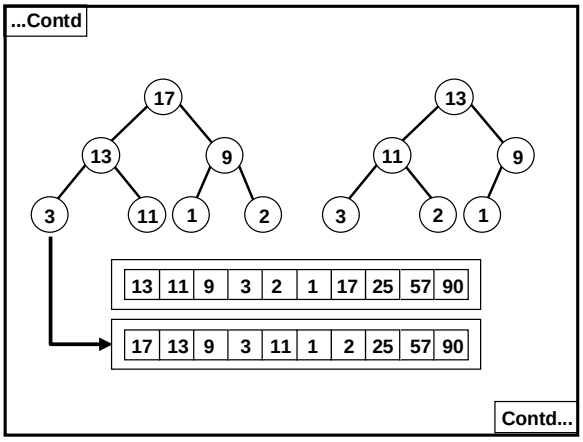


Contd...

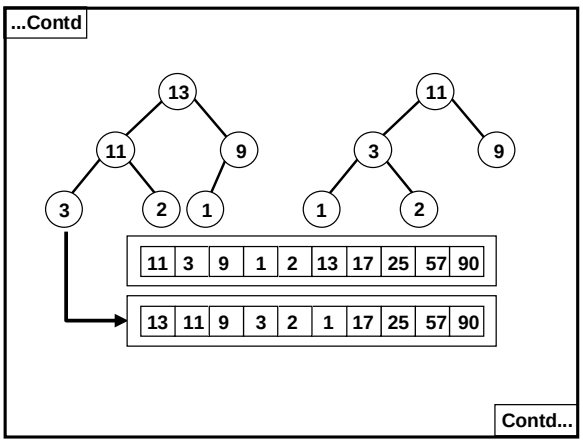
...Contd

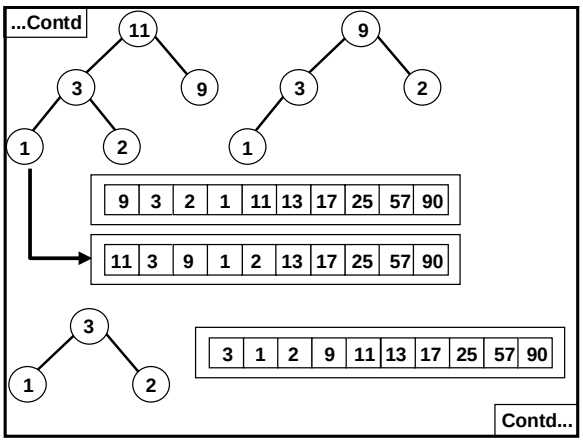


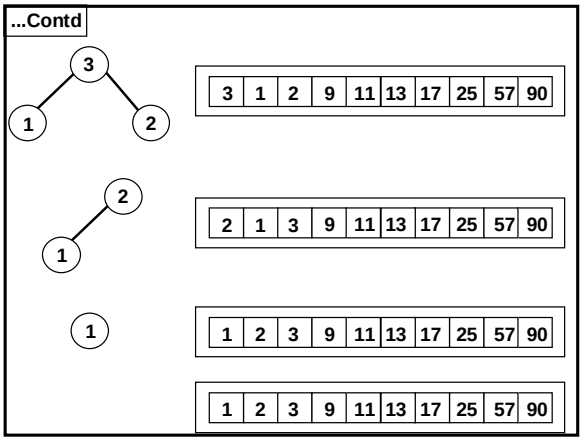
...Contd



...Contd




...Contd
11
3
9
1
2
9
3
2
1
9 3 2 1 11 13 17 25 57 90
11 3 9 1 2 13 17 25 57 90
3
1
2
3 1 2 9 11 13 17 25 57 90
Contd...


...Contd
3
1
2
3 1 2 9 11 13 17 25 57 90
2
1
2 1 3 9 11 13 17 25 57 90
1
1 2 3 9 11 13 17 25 57 90
1 2 3 9 11 13 17 25 57 90

AVL Trees & B-Trees

Yashavant Kanetkar

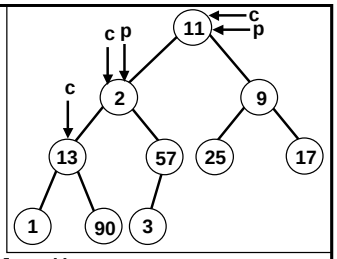
Objectives

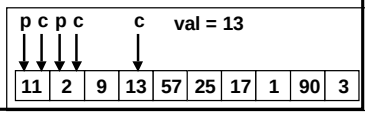
- What are AVL trees
- What are 2 - 3 trees
- What are B - trees

Create Heap

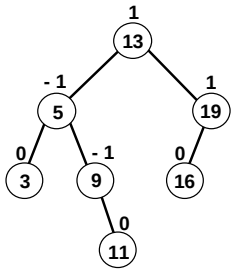
```
# define  MAX  10
main( )
{
    int  arr[ ] = { 11, 2, 9, 13, 57, 25, 17, 1, 90, 3 } ;
    int  i ;
    createheap ( arr, MAX ) ;
    printf ( "Heap: " ) ;
    for ( i = 0 ; i < MAX ; i + + )
        printf ( "%d\t", arr[ i ] ) ;
}
```

```
createheap ( int  *a, int  n )
{
    int  val, i, c, p ;

    for ( i = 1 ; i < n ; i + + )
    {
        val = a[ i ] ;
        c = i ;
        p = ( c - 1 ) / 2 ;

        while ( c > 0 && a[ p ] < val )
        {
            a[ c ] = a[ p ] ;  c = p ;
            p = (c - 1) / 2 ;
        }
        a[ c ] = val ;
    }
}
```





AVL Tree



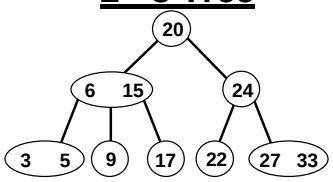
Adelson - Velskii - Landis

Balanced Binary Tree

Bal. Factor = ht. of left ST - ht. of right ST

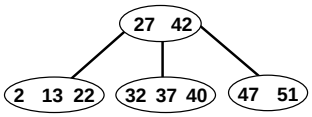
```
struct  AVL
{
    struct  AVL  *lc ;
    int  balfact ;
    int  data ;
    struct  AVL  *rc ;
} ;
```

2 - 3 Tree



- Max. 2 values, min. 1 value
- Non - leaf node must have max. 3 children, min. 2 children
- All the leaf nodes should appear on the same level
- Value(s) at left child < value(s) at middle child
- Value(s) at middle child < value(s) at right child
- If i and j are 2 values on same node then i < j
  - all values on left child < i
  - all values on middle child > i & < j
  - all values on right child > j

B - Tree



Order - 4

- Non - leaf nodes (except root) have min. n / 2 children, max. n children
- Node with n children must have n - 1 values
- All the values of a particular node are in increasing order
- All the leaf nodes should appear on the same level
- Values present on any child between any two values i & j should be > i & < j, where i < j
- Values present on left child must be < 1st value
- Values present on right child must be > last value

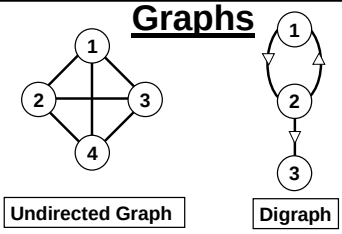
Graphs

Yashavant Kanetkar

Objectives

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

Graphs

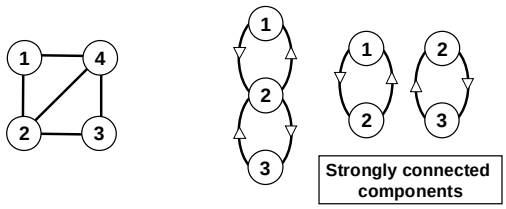


Undirected Graph

Digraph

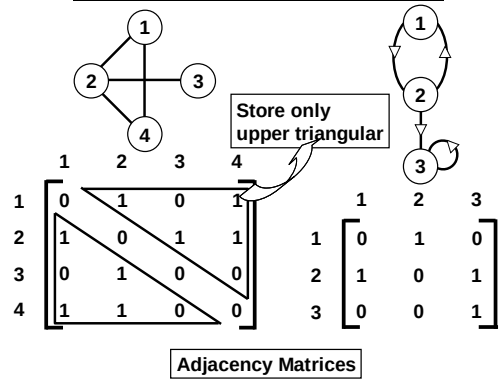
- Graph is set of v & e
  v - vertices (finite & non-empty), e - edges (pair of vertices)
- Undirected graph - edge is unordered, (1, 2) & (2, 1) are same
- Digraph - edge is ordered, <1, 2> & <2, 1> are different
- In <1, 2> 1 is tail and 2 is head
- In undirected graph 1 and 2 are adjacent
- In undirected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

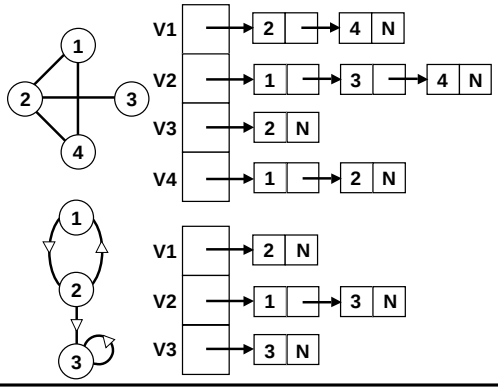


- Path - from 1 to 3 is sequence of vertices 1, 4, 3 with edges (1, 4), (4, 3)
- Simple path - starting & ending vertex distinct. 1, 4, 3
- Cyclic path - starting & ending vertex is same. 1, 2, 4, 1
- Connected - if there is a path from v1 to v2
- Strongly connected - if there is a path from v1 to v2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

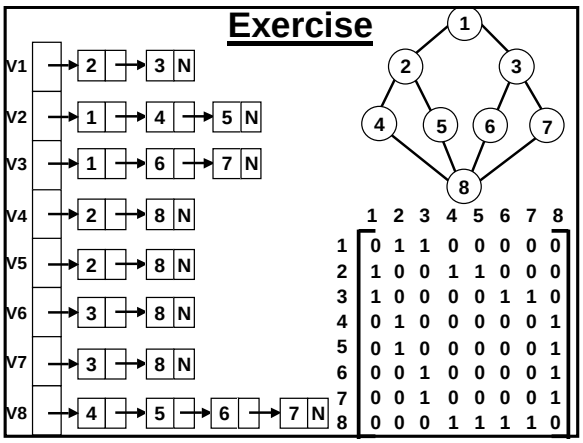
Graph Representation



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

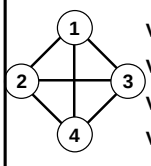
Adjacency Multilist

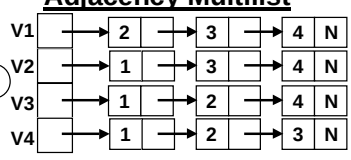
Yashavant Kanetkar

Objectives

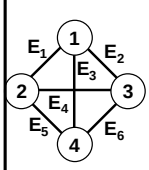
- Adjacency matrices
- Adjacency multilist
- Orthogonal representation of graphs

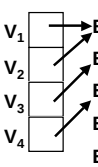
Adjacency Multilist





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





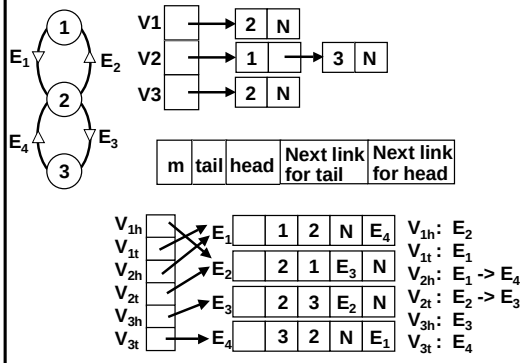
V_1: E_1-> E_2 -> E_3

V_2: E_1-> E_4 -> E_5

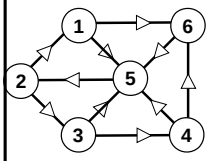
V_3: E_2-> E_4 -> E_6

V_4: E_3-> E_5 -> E_6

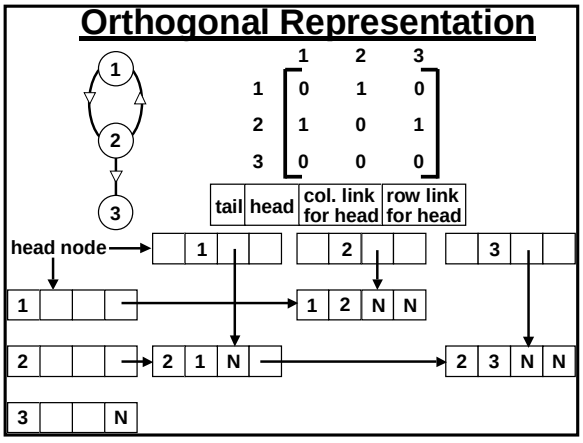
Adjacency Multilist



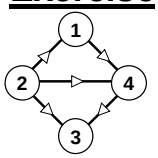
Exercise



Orthogonal Representation

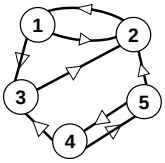


Exercise



Exercise

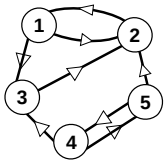
Give all graphical representations for the following graph

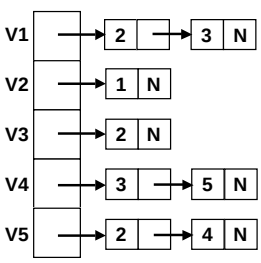


	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	1	0	1	0

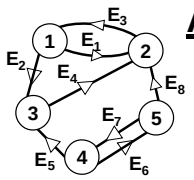
Adjacency Matrices

Adjacency List

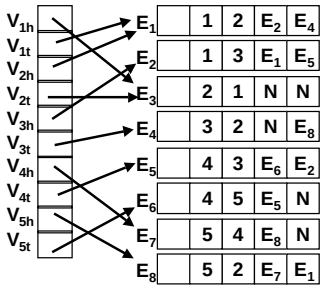




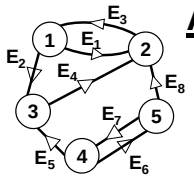
Adjacency Multilist



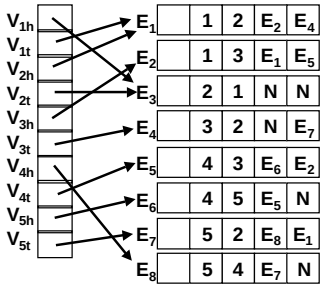
V_{1h}: E_3
V_{1t}: E_1 -> E_2
V_{2h}: E_1 -> E_4 -> E_8
V_{2t}: E_3
V_{3h}: E_2 -> E_5
V_{3t}: E_4
V_{4h}: E_7
V_{4t}: E_5 - > E_6
V_{5h}: E_6
V_{5t}: E_7 -> E_8



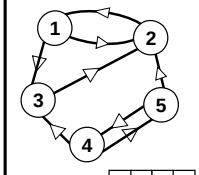
Adjacency Multilist



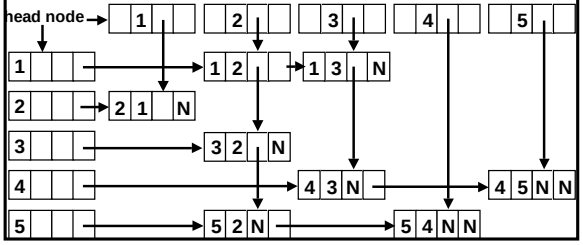
V_{1h}: E_3
V_{1t}: E_1 -> E_2
V_{2h}: E_1 -> E_4 -> E_7
V_{2t}: E_3
V_{3h}: E_2 -> E_5
V_{3t}: E_4
V_{4h}: E_8
V_{4t}: E_5 - > E_6
V_{5h}: E_6
V_{5t}: E_7 -> E_8



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

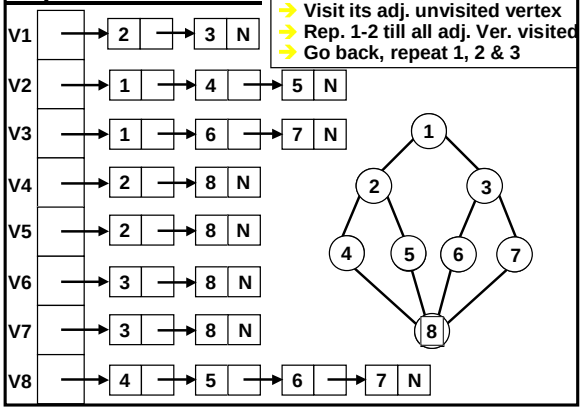
Yashavant Kanetkar

Objectives

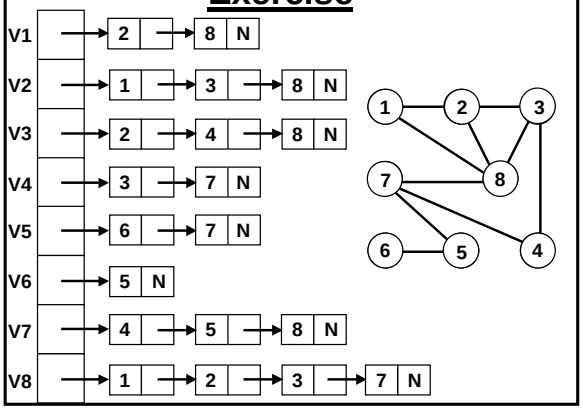
- Study Depth First Search

Depth First Search

- 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 "alloc.h"
#define  MAX  8
struct  node
{
     int  data ;
     struct  node  *next ;
} ;
struct  node  * add ( int ) ;
main( )
{
     struct  node  *arr[ MAX ] ;
     struct  node  *v1, *v2, *v3 ;

     v1 = add ( 2 ) ;
     v2 = add ( 3 ) ;
     arr[ 0 ] = v1 ;
     v1 -> next = v2 ;

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

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

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

Contd...

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

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

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

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

     arr[ 7 ] = v1 ;
     v1 -> next = v2 ;
     v2 -> next = v3 ;
     v3 -> next = v4 ;

     dfs ( 1, arr, visited ) ;

     for ( i = 0 ; i < MAX ; i++ )
          del ( arr[ i ] ) ;
}
```

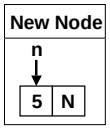
struct node *v4

int visited[MAX] = { 0 } ;

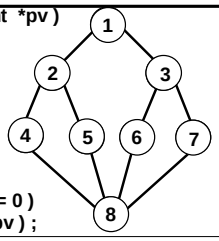
int i

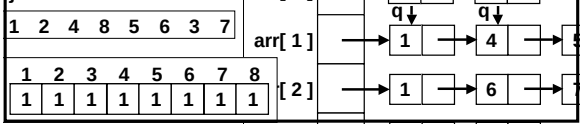
add( )

```
struct  node * add ( int  val )
{
     struct  node  *n ;
     n = ( struct  node * ) malloc ( sizeof ( struct  node ) ) ;
     n -> data = val ;
     n -> next = NULL ;
     return  n ;
}
```



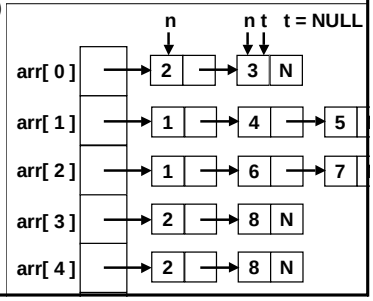

```
dfs ( int  v, struct  node  **p, int  *pv )
{
     struct  node  *q ;
     pv[ v - 1 ] = 1 ;
     printf ( "%d\t", v ) ;
     q = * ( p + v - 1 ) ;
     while ( q != NULL )
     {
          if ( pv[ q -> data - 1 ] == 0 )
               dfs ( q -> data, p, pv ) ;
          else
               q = q -> next ;
     }
}
```





del( )

```
del ( struct  node  *n )
{
     struct  node  *t ;
     while ( n != NULL )
     {
          t = n -> next ;
          free ( n ) ;
          n = t ;
     }
}
```



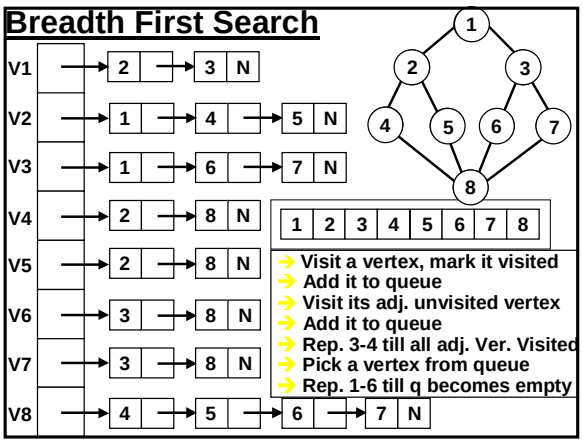
Breadth First Search

Yashavant Kanetkar

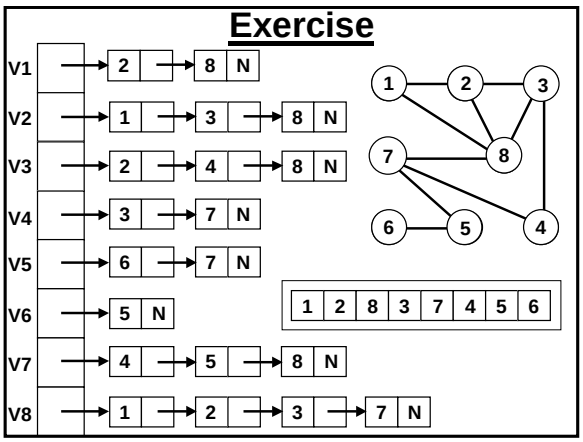
Objectives

- Study Breadth First Search

Breadth First Search



Exercise




```
bfs ( int  v, struct  node  **p, int  *pv )
{
    struct node *u ;
    struct  queue  q ;
    q.f = q.r = -1 ;
    pv[ v - 1 ] = 1 ;
    printf ( "%d\t", v ) ;
    addqueue ( &q, v ) ;
    while ( isempty ( &q ) == 0 )
    {
        v = delqueue ( &q ) ;
        u = * ( p + v - 1 ) ;
```

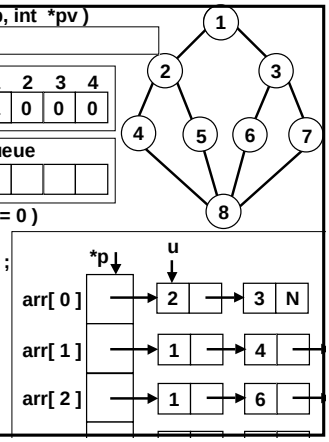
1

1	2	3	4
1	0	0	0

queue: 1

```
struct  queue
{
    int  q[ MAX ] ;
    int  f, r ;
} ;
```

Contd...

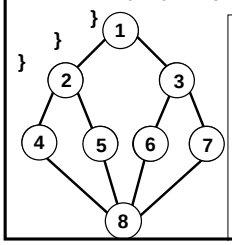


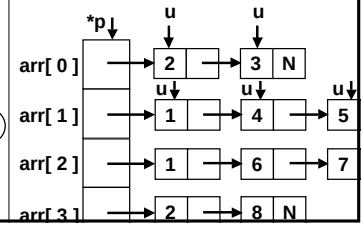

```
...Contd  while ( u != NULL )
          {
              if ( pv [ u -> data - 1 ] == 0 )
              {
                  addqueue ( &q, u -> data ) ;
                  pv [ u -> data - 1 ] = 1 ;
                  printf ( "%d\t", u -> data ) ;
              }
              u = u -> next ;
          }
    }
}
```

1 2 3 4 5 6 7 8

queue: 2 3 4 5

1 1 1 1 1 1 1 1





addqueue( )

```
addqueue ( struct  queue  *pq, int vertex )
{
     if ( pq -> r == MAX - 1 )
     {
          printf ( "\Full." ) ;
          exit( ) ;
     }

     pq -> r + + ;
     pq -> q[ pq -> r ] = vertex ;

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

delqueue( ) & isempty( )

```
delqueue ( struct  queue  *pq )
{
    int  data ;
    if ( pq -> f == - 1 )
    {
         printf ( "\nEmpty." ) ;
         exit( ) ;
    }
    data = pq -> q[ pq -> f ] ;
    if ( pq -> f == pq -> r )
         pq -> f = pq -> r = -1 ;
    else
         pq -> f + + ;
    return  data ;
}
```

```
isempty ( struct  queue  *pq )
{
     if ( pq -> f == - 1 )
          return  1 ;
     return  0 ;
}
```

Spanning Tree

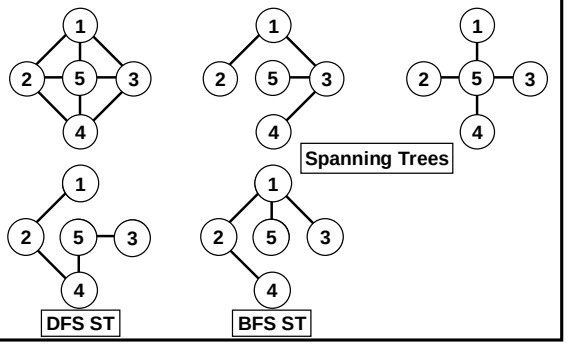
Yashavant Kanetkar

Objectives

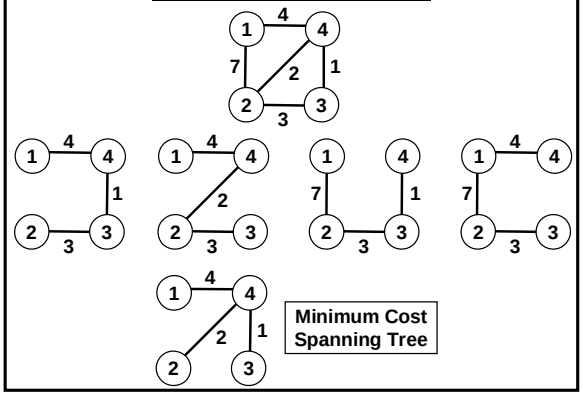
- Spanning Tree
- Cost of Spanning Tree
- Kruskal's Algorithm to determine minimum cost Spanning Tree

Spanning Tree

- Undirected tree
- Contains edges necessary to connect all vertices

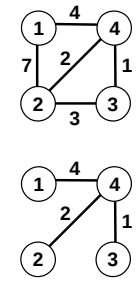


Calculating Cost



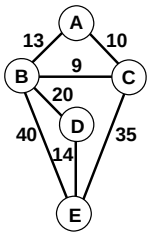
Kruskal's Algo

- 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

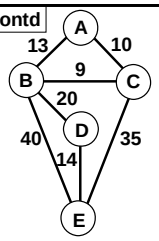
Exercise

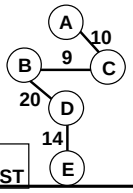


Edge	Cost	Tree	Action

Contd...

...Contd





Min. Cost ST

Edge	Cost	Tree	Action
		A B C D E	
		A B C D E	

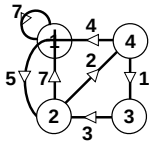
Dijkstra's Algorithm

Yashavant Kanetkar

Objectives

- Dijkstra's Algorithm to determine minimum cost Spanning Tree
- AOV Network
- Topological Order
- AOE Network

Dijkstra's Algorithm

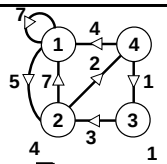


	1	2	3	4
1	7	5	∞	∞
2	7	∞	∞	2
3	∞	3	∞	∞
4	4	∞	1	∞

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

Contd...

...Contd



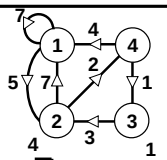
	1	2	3	4
1	7	5	∞	∞
2	7	∞	∞	2
3	∞	3	∞	∞
4	4	∞	1	∞

	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



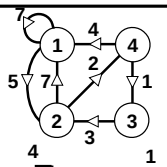
	1	2	3	4
1	7	5	∞	∞
2	7	12	∞	2
3	∞	3	∞	∞
4	4	9	1	∞

	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



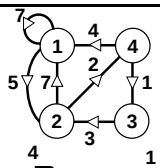
	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	9	1	11

	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



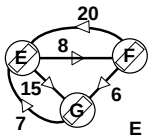
	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	4	1	6

	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	∞	∞

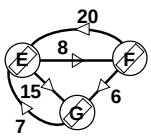
	E	F	G
E	-	EF	EG
F	FE	-	FG
G	GE	-	-

	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

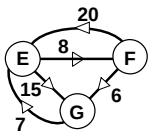
	E	F	G
E	-	EF	EG
F	FE	FEF	FG
G	GE	GEF	GEG

	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

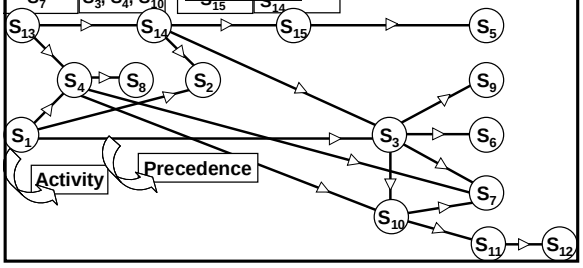
Sub. No.	Prereq.
~~S_1~~	~~None~~
~~S_2~~	~~S_1, S_{14}~~
~~S_3~~	~~S_1, S_{14}~~
~~S_4~~	~~S_1, S_{13}~~
~~S_5~~	~~S_{15}~~
~~S_6~~	~~S_3~~
~~S_7~~	~~S_3, S_4, S_{10}~~

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

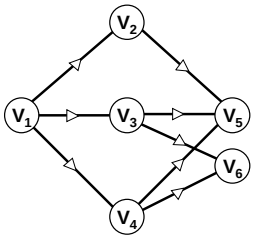
Activity On Vertex Network

V - Activity / Task
E - Precedence Relation



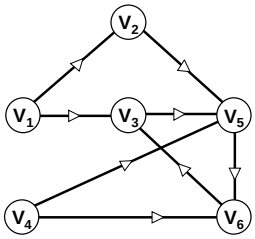
Exercise - I

Vertex	Prerequisite
V_1	
V_2	
V_3	
V_4	
V_5	
V_6	



Exercise - II

Vertex	Prerequisite
V_1	
V_2	
V_3	
V_4	
V_5	
V_6	



Not Feasible

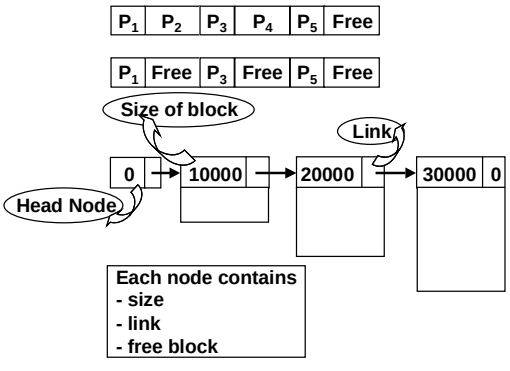
Memory Management-I

Yashavant Kanetkar

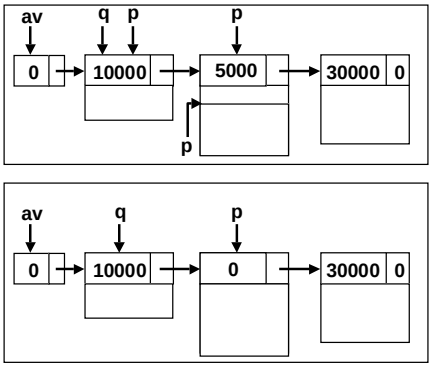
Objectives

- Study Memory Management
- Study First Fit Algorithm
- Study Best Fit Algorithm

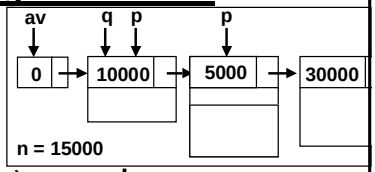
Memory Management



First Fit



Algo. - First Fit




```
firstfit ( *av, n, *p )
{
   p = q = link ( av ) ;
   while ( p != 0 )
   {
      if ( size ( p ) >= n )
      {
          setsize ( p, size ( p ) - n ) ;
          if ( size ( p ) == 0 )
             setlink ( q, link ( p ) ) ;
          else
                p = p + size ( p ) ;
          break ;
      }
          q = p ;
          p = link ( p ) ;
      }
}
```

Memory Management-II

Yashavant Kanetkar

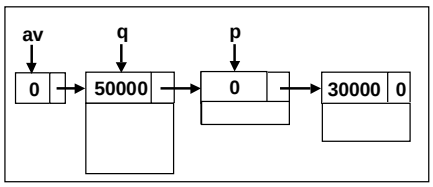
Objectives

- Study Next Fit Algorithm

Best Fit

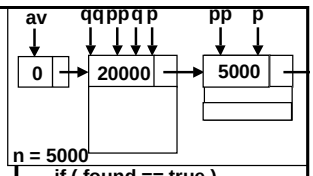
Disadv. of firstfit Algo.

- Tendency to create too many small nodes
- Search always starts at beginning of available list



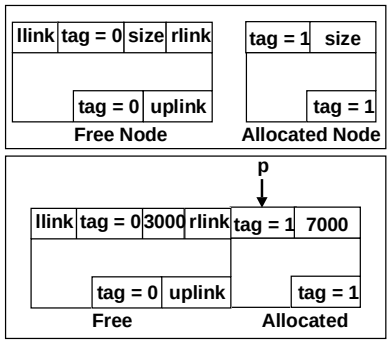
Algo. - Best Fit

```
bestfit ( *av, n, *p )
{
    p = q = link ( av ) ;
    diff = 100000 ;
    found = false ;
    while ( p != 0 )
    {
        if ( size ( p ) >= n )
        {
            d = ( size ( p ) - n ) ;
            if ( d < diff )
            {
                pp = p ;  qq = q ;
                found = true ;
                diff = d ;
            }
        }
        q = p ;  p = link ( p ) ;
    }
```

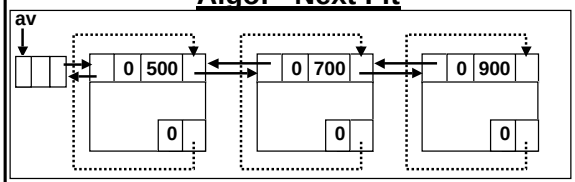



```
    if ( found == true )
    {
        q = qq ;  p = pp ;
        setsize ( p, size( p ) - n ) ;
        if ( size ( p ) == 0 )
            setlink ( q, link ( p ) ) ;
        else
            p = p + size ( p ) ;
    }
    else
        printf ( "Insuff. mem." );
}
```

Next Fit



Algo. - Next Fit

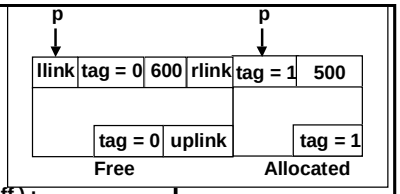



```
nextfit ( *av, n, p )
{
    p = rlink ( av ) ; alpha = 50 ;
    while ( 1 )
    {
        if ( size ( p ) >= n )
        {
            diff = ( size ( p ) - n ) ;
            if ( diff < alpha )
            {
```

```
            q = llink ( p ) ;  r = rlink ( p ) ;
            setlink ( rlink ( q ), r ) ;
            setlink ( link ( r ), q) ;
            settag ( p, 1 ) ;
            settag ( p + size ( p ) - 1, 1) ;
            av = llink ( p ) ;
        }
```

Contd...

...Contd




```
    else
    {
        setsize ( p, diff ) ;
        setuplink ( p + size ( p ) - 1, p ) ;
        settag ( p + diff - 2, 0 ) ;
        av = p ;
        p = p + diff ;
        setsize ( p, n ) ;
        settag ( p, 1 ) ;
        settag ( p + n - 1, 1 ) ;
    }
    break ;
}
```

```
        p = rlink ( p ) ;
        if ( p == rlink ( av ) )
            break ;
    }
    p = 0 ;
}
```

Garbage Collection

Yashavant Kanetkar

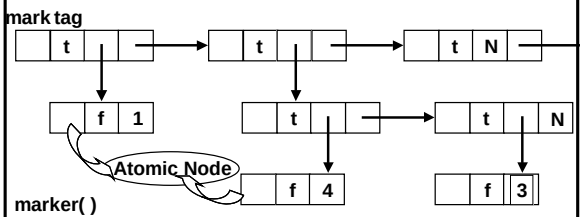
Objectives

- Study Garbage Collection and Compaction

Garbage Collection

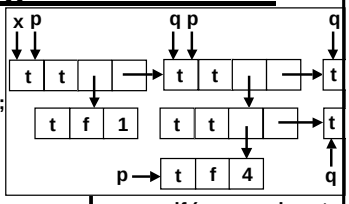
- Method of detecting & reclaiming free nodes
- Done in two phases:
  - Marking Phase - marks all nodes that are accessible from an external pointer (nodes in use)
  - Collection Phase - free all nodes that are not marked

Marking Phase



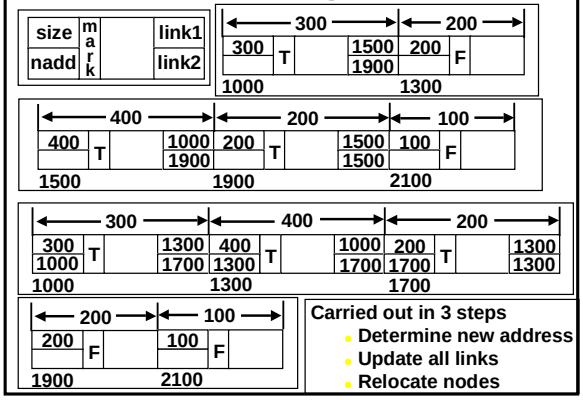

```
marker( )
{
    for each pointer x such that x != NULL
    {
        x -> mark = t ;
        if ( x -> tag == t )
            mark1 ( x ) ;
    }
}
```

Algorithm Of Mark




```
mark1 ( *x )
{
  top = 0 ;
  push ( stack, top, x ) ;
  while ( top > 0 )
  {
      pop ( stack, top, p ) ;
      while ( 1 )
      {
          q = p -> link ;
          if ( q != NULL )
          {
              if ( q -> tag == t &&
                      q -> mark == f )
              push ( stack, top, q ) ;
              q -> mark = t ;
          }
          p = p -> dlink ;
          if ( p == NULL )
              break ;
          if ( p -> mark == t
              || p -> tag == f )
              break ;
          p -> mark = t ;
      }
      p -> mark = t ;
  }
}
```

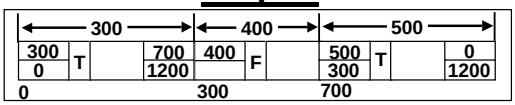
GC & Compaction

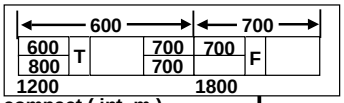


Carried out in 3 steps

- Determine new address
- Update all links
- Relocate nodes

Step - I

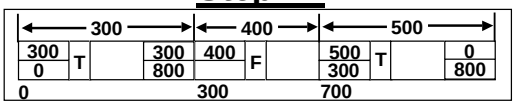


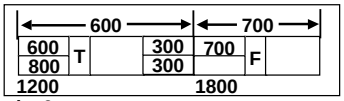



```
compact ( int  m )
{
     i = av = 0 ;
     while ( i < m )
     {
        if ( mark ( i ) )
        {
            newadd ( i ) = av ;
            av = av + size ( i ) ;
        }
        i = i + size ( i ) ;
     }
```

i	av
0	0
300	300
700	800
1200	1200
1800	1200
2500	1200

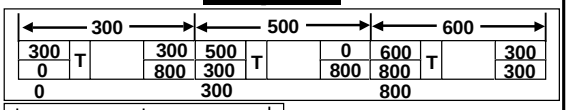
Step - II

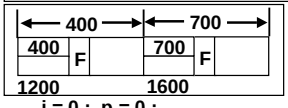





```
i = 0 ;
while ( i < m )
{
   if ( mark ( i ) )
   {
      link1 ( i ) = newadd ( link1 ( i ) ) ;
      link2 ( i ) = newadd ( link2 ( i ) ) ;
   }
   i = i + size ( i ) ;
}
```

Step - III






```
     i = 0 ;  p = 0 ;
     while ( i < m )
     {
        if ( mark ( i ) )
        {
            for ( j = i ; j < i + size ( i ) - 1 ; j + +, p + + )
                mem[ p ] = mem [ j ] ;
        }
        i = i + size ( i ) ;
     }
}
```

File Organization

Yashavant Kanetkar

Objectives

- Study File Organization

Files

- Contains records
- Records contain one or more fields
- Each field represents an item of information
- Purpose of file organization - Easy Retrieval and Updation
- Retrieval using Primary or Secondary keys
- Mode of Retrieval - Real time and Batched
- Mode of Update - Real time or Batched

File Organization

- Representing records on external storage medium
- Sequential:
  - New record inserted at end of file
  - Records might be of fixed or variable length
  - Insertion and Deletion is inefficient
  - Only Primary key ordering
  - Efficient for batch processing
- Random:
  - Records stored at random locations
  - Methods of access:
    - Direct Addressing - Space inefficient
    - Hashing - Easy Insertion and Deletion
      - Multiple keys can be used in f( )
    - Directory lookup - Indexed sequential access

- Linked Organization:
  - Add. of next record is kept in current record
  - Easy insertion and deletion
  - Inefficient - As search is sequential
  - To improve efficiency - Range based index
- Inverted:
  - Multiple indexes
  - Record offsets matching an index is stored with index itself

Ind. Seq. Access

```
struct  emp
{
     int  id ;
     char  name[ 20 ] ;
     int  age ;
     float  sal ;
} ;
```

ID	Rel. Off.
121	0
125	28
145	84
210	56
215	224
218	252
220	112
234	168
245	140
266	196

ID	Name	Age	Salary
121	Vishal	32	12000
125	Saurabh	24	4500
210	Sunil	24	5350
145	Shirish	22	4560
220	Shekhar	26	5670
245	Hetal	33	8790
234	Vipin	32	6780
266	Ashwini	22	3450
215	Sonal	21	3445
218	Shweta	28	8890