
# KTU NOTES

The learning companion.

**KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS**

# Data structure and algorithms

→ System life cycle.

- series of stages ~~that~~ in the development of a new information system.

4 has 5 phases

- Requirment
- analysis
- design
- coding
- verification.



Requirment → details about the input and output

analysis → Break the problem down into manageable pieces. Top-down approach. bottom-down

Design → Two perspectives

* Data objects perspective.
* Operations perspective.

-- specification of algorithm (abt algorithm)

Refinement and coding → choose representation for data objects

. algorithms.

. must be in a crct order.

algorithm ⟷ programs

Verification → developing correctness, testing the program, and removing the error.

19-8-20
Wednesday.

## Algorithm.

- finite set of instructions that followed in a particular task.

(1) input.
→ Zero or more quantity.

(2) output
at least one

(3) Definiteness
Each instruction is clear and unambiguous.

(4) finiteness.
The algorithm terminates after a finite number of steps.

(5) Effectiveness.

## performance analysis

- There may be more than one way to solve a problem
- choose the best one
- must consider time, space complexity
- Time complexity - Time taken. , space → space taken.
- Time and space complexity depends on hardware, OS, processors, etc.
- we usually consider the execution time of an algorithm

## measuring time of an algorithm

- experimental study.
- Run the program
- use method like System.currentTimeMillis() to get accurate value.

→ uses of high-level description of the algorithms.

- primitive operations:

22-8-20
saturday

## Time and space complexity.

analysing an algorithm ⟹ determining the amount of resource, ~~space and~~ time and memory to execute it.

So that size and space complexity can be stated as a function of input.

space needed depends on:

fixed part: varies from prg → prgm.

- space needed for storing instructions, constants, variable.

variable part

: space needed for recursion stack and for structured variables that are allocated space dynamically during the run time.

- Best-case running time

: needed variable comes first (min)

- worst-case "

needed ⟶ last (max)

- averag-case "

needed ⟶ middle or nearby (medium)

- Time and space complexity can be expressed using a function $f(n)$ where $n$ is the input size.

efficiency ~~can be~~ of the algorith or running time can be given as the number of instructions it contains

(linear program)

eg: $s = a + b$ here only one instruction is there

- a constant time occur in a linear program.

If loop is there, then the efficiency may be depends on the number of loops and running time of each loop.

Linear loops.

- at first determine the number of times the statement in the loop will be executed.

- loop execution ∝ complexity.

eg:- for (i=0; i<100; i++)

Here 100 is the loop factor. $f(n) = n$

for (i=0; i<n; i+=2)

Here n/2 is the loop factor. $f(n) = n/2$

100 $i = 1/2$

Logarithmic loops.

eg:

for (i=1; i<1000; i*=2)

loop factor 10 times ⟶ $2^9 + 1$

512.

so here

$f(n) = \log n$

means power ⟶ log.

## Nested loops.

### a) Linear & logarithmic loop.

```
for(i=0; i<10; i++)
    for(j=1; j<10; j*=2)
        Statement block.
```

∴ here $f(n) = n \log n$

### Quadratic loop

```
for(i=0; i<10; i++)
  for(j=0; j<10; j++)
```

$f(n) = n \times n = n^2$

### Dependent quadratic loop.

```
for(i=0; i<10; i++)
    for(j=0; j<=i; j++)
```

here

$$f(n) = \frac{n \times (n+1)}{2}.$$

4-8-20

## Time complexity

```
for i=1 to n do ——————— n+1
    for j=1 to n do
        k=k+1 ] (2n+1)n times.
```

~~n times~~

Total number of instructions = $n+1+n(2n+1)$

eg: for i=1 to n

Sum = sum + i ——————— n times.

so it works $n+1$ time

n → right

1 → wrong.

i.e

i=1 to n ——————→ $n+1$ times

Then

```
j=1 to n (n+1)
k=k+1 ——→ (n)
```

} $2n+1$ times.

Then This is inside the $I^{st}$ loop so.

It work $n(2n+1)$ times, since the $I^{st}$ loop works n times

∴ Total $n+1+n(2n+1)$

$= n+1+2n^2+n$

$= 2n^2+2n+1$.

$= n+1 + n(n+1) + n \times n$

$= n+1 + n^2 + n + n^2$

$= 2n^2 + 2n + 1 \Longrightarrow f(n)$

## Asymptotic notations

They are mathematical notation used to describe the running time

mainly theta notation, Omega notation and Big-O notation.

$2n+1 \leq 3n$.

- $2n+1$

$n=1 \qquad 2\times1+1 = 2+1$

29-8-20

- In bubble sort, if input array is already sorted then it take less time

: best case.

- when it is in reverse order

worse case.

- Normall

medium case.

# Big-oh notation

worst case

$f(n) = O(g(n))$

$f(n) \leq c * g(n)$ for all $n \geq n_0$, $c > 0$

It gives the upper bound (worst case) on running time.

eg: $3n+2 \leq 4n \quad c = 4 \; \forall \; n \geq 2$

$\Rightarrow 3n+2 = O(n)$

let's assume $g(n) = n$

To find constant $c$.

$3n+2 \leq 10n \quad c = 10$

This equation is true always from $n = 1$

$n=0 \Rightarrow 2 \leq 0$ ✗

$n=1 \Rightarrow 5 \leq 10$ ✓

$n=2 \Rightarrow 8 \leq 20$ ✓

* $3n+2 \leq 10n$ for all $n \geq 1$, $c = 10$

complexity is Big-oh $n$

To check

$3n+2 \leq 2n^2 \quad c = 2$

here the equation is true from $n = 2$

$n=0 \Rightarrow 2 \leq 0$ ✗

$n=1 \Rightarrow 5 \leq 2$ ✗

$n=2 \Rightarrow 8 \leq 8$ ✓

$n=3 \Rightarrow 11 \leq 18$ ✓

also all $3n+2n = 5n$ take

best one having shorter no value.

$1 < \log n < n < n\log n < n^c < c^n < n!$

- $O(1)$
- $O(\log n)$
- $O(n^2)$
- $O(n^3)$
- $O(n\log n)$
- $O(2^n)$

} probable one.

7-9-20
monday.

## Asymptotic notation

Mathematical notation used to describe the running time of an algorithm.

$2n$ $n^2$.

for $n=2$ ✓ for $n=1$ ✓, $n=2$ ✓

$2n$ is better than $n^2$ when $n \geq 2$

- $3n+2 \leq 4n$ where $C=4$ // .

| | |
|---|---|
| $n=0$ | $2 \leq 0$ ✗ |
| $n=1$ | $5 \leq 4$ ✗ |
| $n=2$ | $8 \leq 8$ ✓ |
| $n=3$ | $11 \leq 12$ ✓ |

$3n+2 \leq 4n$ only when $n \geq 2$

for a better result add $3+2=5$ ie $3n+2 \leq 5n$.
ie $C=5$ ($n=1$ $5 \leq 5$)

- $3n+2 \leq 2n^2$

| | |
|---|---|
| $n=0$ | $2 \leq 0$ ✗ |
| $n=1$ | $5 \leq 2$ ✗ |
| $n=2$ | $8 \leq 8$ ✓ |

$3n+2 \leq 2n^2$ is true only when $n \geq 2$

ie $n_0 = 2$ $C=2$.

So the complexity can be represented as

- Big-O gives the worst case.

## Big-omega Notation

$f(n) = \Omega(g(n))$

$f(n) \geq c * g(n)$ for all $n \geq n_0$ $c > 0$

eg:

~~$3n+2 \geq 3n$~~ $n_0 = 1$ $c = 3$

$3n+2 \geq 3n$

$n=0$ $2 \geq 0$ ✓

$n=1$ $5 \geq 3$ ✓

$n=2$ $8 \geq 6$ ✓

~~$n(n)$ $\Omega(n)$ is $3n$~~

omega Notation gives the lower bound.

Its complexity ~~$\Omega(3n)$~~

$\Omega(n)$ is the complexity.

## Theta Notation

-average running time.

$f(n) = \Theta(g(n))$

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$, $c > 0$.

$f(n) = 3n+2$

$3n \leq 3n+2 \leq 4n$. $n \geq 2$ $c_1 = 3$. $c_2 = 4$.

average $\Theta(n)$

6-9-20

Class room

Q-I

$f(n) = 2n + 2m + 1$

~~$f(n) = O(n+m)$~~

Big-oh of $(n+m)$ is correct or

Big-oh of $\max(n, m)$.

i.e $= O(n+m)$.

~~Q-2.~~

```
int a=0   b=0          ——— (1)
for (i=0 ; i<n ; i++)  ——— n+1
[
   a = a+rand();       ——— n
}
for (j=0 ; j<m ; j++)  ——— m+1
{                      ——— m
   b = b+rand()
}
```

So $f(n) = 2n + 2m + 3$ So all the coefficients are neglected.

Q-2.

```
int i=1;               ——— (1)
while (i<=n)           ——— (log n)
{
   printf("*")         ——— n
   i = 2*i;
}
```

$= O(\log n)$

$f(n) =$

G find the big-o time complexity

Ans

```
int a=0, b=0;                    ——— (1)
for (i=0; i<n; i++)              ——— n+1
{
    a=a+rand();                  ——— n
}
for (j=0; j<m; j++)              ——— m+1
{
    b=b+rand();                  ——— m
}
```

$\therefore$

$f(n) = 1 + n+1 + n + m+1 + m$

$= 2n + 2m + 3$

$2n+2m+3 \leq 3(m+n)$

n=0 m=0 3 ≤ 9 ✗

n=1 4 ≤ 6

~~where c=3~~

i.e

Big-Oh-Notation is $O(n+m)$

$= O(\max(n, m))$

b.

```
int i = 1;                    ——— (1)
while (i<=n)                  ——— logn + 1
{
    printf ("\n");            ——— logn
    i = 2 * i;                ——— logn
}
```

$f(n) = 1 + \log n + 1 + \log n + \log n$

$= 3\log n + 2$.

ie Big-O notation is $O(\log n)$.

c.

```
int a=0;                      ——— 1
for (i=0; i<N; i++)           ——— N+1
{
    for (j=N; j>i; j--)       ——— N+1, N, N-1, N-2, ..... 0
    {                                 = N(N+1)/2
        a = a + i + j;        ———
    }
}
```

here $n = N+1$

$\frac{N(N+1)}{2}$ $\quad = \frac{(N+1)(N+2)}{2}$.

$f(n) = \frac{N(N+1)}{2} + \frac{(N+1)(N+2)}{2} + (N+1) + 1$ — no need of this, only basic count is needed.

$= \frac{N^2+N}{2} \quad = \frac{N^2}{2} + \frac{N}{2}$

Big-O notation

Time complexity $= O(N^2)$

$N^2 + N$ since $N^2$ is taken so it shows more complexity.

d.

```
int i=n;                                  ——→ 1
while (i>0)                               ——→ n+1   = 1+logn
{
  for (int j=0; j<n; j++)                 ——→ logn
    printf("*");                          ——→ logn
    i=i/2;                                ——→ logn.
}
```

~~1+n+1~~

$f(n) = 1 + 1 + n\,[\log n + \log n + \log n]$

$= 1 + 1 + n\,[3\log n]$

$= 2 + 3n\log n.$

∴ Time complexity = $O(n \log n)$

Note:

——→ 1

logn + 1

(n+1)(logn)

n(logn)

logn

6. linear search complexity (lap – $S_3$ . linearsearch)

```
max = a[0];                                   ——— (1)
for i=1 to n                                  ——— n+1
    if (a[i] > max))                          ——— n
        max = a[i];                           ——— n
    return max;                               ——— 1
```

$1+n+n+n+1 = 3+3n = O(n)$

12-9-20.

# Data structure.

- representation of logical relationship existing b/w individual.
- program = Algorithm + Data structure.

non-primitive data structure.

- create
- selection
- updating
- searching
- sorting
- merging
- Delete.

→ Arrays

set of index and value.

- Data structure

for each index, there is a value associated with that index.

- array is set of pair and index

polynomial rep using array.

$X_1 = 3x^1 + 5x^2 + 7x^3$

$X_1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

| i=0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 3 | 5 | 7 |

$X_2 = 10x^0 + 3x^1 + 5x^2$

$X_2 = 10x^2 + 3x^1 + 5x^2 + 0x^3$

| j=0 | 1 | 2 | 3 |
|---|---|---|---|
| 10 | 3 | 5 | 0 |

To calculate sum

$X_3 = X_1 + X_2$

$= 10x^0 + 6x^1 + 10x^2 + 7x^3$

| k=0 | 1 | 2 | 3 |
|---|---|---|---|
| 10 | 6 | 10 | 7 |



$X_1 = 7x^4 + 5x^2 + 3x^1$

$X_2 = 5x^3 + 3x^1 - 8x^0$

$X_1 =$

| | i=0 | 1 | 2 |
|---|---|---|---|
| C | 4 | 5 | 3 |
| E | 4 | 2 | 1 |

$X_2 =$

| | j=0 | 1 | 2 |
|---|---|---|---|
| C | 5 | 3 | -8 |
| E | 3 | 1 | 0 |

$X_3 = 4x^4 + 5x^3 + 5x^2 + 3x^1 - 8x^0 + 3x^{-1}$

$X_3 =$

| | k=0 | | | | | |
|---|---|---|---|---|---|---|
| C | | | | | | |
| E | | | | | | |

Youtube tutorial portion:

① $p(x) = 3x^5 + 2x^4 + 5x^2 + 2x + 7$

exp

$p(x) = 3x^5 + 2x^4 + 5x^2 + 2x + 7$

coefficient

$n = 5$.

| coe | 3 | 2 | 5 | 2 | 7 |
|---|---|---|---|---|---|
| ex | 5 | 4 | 2 | 1 | 0 |

for c program.

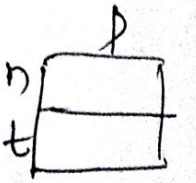
```
struct Term
{
    int coeff;
    int expo;
};
```

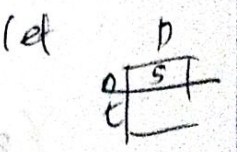
```
struct poly
{
    int n;
    struct term *t;
};
```



---

```
struct poly p;
printf("No of non-zero term");
scanf("%d", &p.n);
p.t = new Term[p.n];
print("Enter polynomial Terms");
```




→ so an array of size 5 will created.

```
for(i=0; i<p.n; i++)
{
  printf("Term no> %d", i++);
  scanf("%d %d", &p.t[i].coeff, &p.t[i].exp);
```

(2) ~~using floating point array.~~

(q-a)

case I: If the exponent of the term pointed by j is $x_2$ is less than that of i in A, then input the value of ~~B~~ i and then increment i and make j current i.e

case II: j is $x_2$ > i is $x_1$

- then copy the j is k and then advance pointer j and k to the next term.

case III: If both are equal, find out the sum increment i, j, k.

Complexity .pdf -2 .

1.

int j=0 ——————(1)

——————→ n+1

——————→ n

——————→ ~~n Cable~~ (n+1)n (while).

j++ ——————→ $n^2$

i.e $1+n+1+n+n(n+1)+n^2 = 2+3n+n^2+n^2 = 2+3n+2n^2$

$f(n) = 2+3n+2n^2$ ← frequency compute.

2. Complexity = $O(n^2)$

$2+3n+2n^2 \leq 4n^2$ $C=4$

where $n_0 = 2$.

$n=0$ $0 \leq 0$ ✗

$n=1$ $7 \leq 9$ ✗

$n=2$ $16 \leq 16$ ✓

$n=3$

2.

(1)

(n+1)

while. (n x n) ←

$n^2$.

i.e. $1+n+1+n^2+n^2 = 2+n+2n^2$.

$f(n) = 2n^2+2+n$.

$2n^2+n+2 \leq 4n^2$. where $C=4$ $n_0=2$.

$n=0$ $2 \leq 0$ ✗

$n=1$ $5 \leq 4$ ✗

$n=2$ $10 \leq 16$ ✓

Complexity = $O(n^2)$

# Sparse matrix.

- If a matrix contains more number of non-zero values are called sparse matrix.
  - It contains very few non-zero element.
  - It waste a lot of space.

eg:

5X6 matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

| Rows | Column | values |
|---|---|---|
| 5 | 6 | 6 |
| 0 | 4 | 9 |
| 1 | 1 | 8 |
| 2 | 0 | 4 |
| 2 | 3 | 2 |
| 3 | 5 | 5 |
| 4 | 2 | 2 |

⇒ sparse matrix rep.

We use sparse matrix, because.

- less storage. (memory storage, less)
- less computing time.

2. (1)

n+1

n

n

$1+n+1+n+n = 2+3n$.

complexity $= O(n)$.

3. 1

1

$\log n$

$\log n$

Tuple → single row in the table in which datas are categorised.

16-9-20.

## Abstract Data type:

ADT is a type for objects whose behaviour is defined by a set of value and a set of operations.

- Abstract: implementation. independent view
- Black box which hides the inner structure and design of the data type.
- ADT's
  - LIST
  - STACK
  - QUEUE

## Stack.

LIFO - Last in first out. [eg: stack of dosa]

FILO (first in last out)

## fundamental operations.

push: Equivalent to an insert.

pop: Deletes the most recently inserted element.

Top: Examines the most recently inserted element.

push and pop are the major ones.

Stack is a linear data structure in which all insertion and deletions are made at one end called top of the stack.

## push

Algorithm.

```
PUSH (stack, size, top, item)
Step 1: If (top == size-1)
            print overflow
        otherwise (step 2)
Step 2: top = top + 1
Step 3: stack[top] = item
Step 4: exit.
```

```
Pop (stack size, top, item)
Step 1: If (top =< 0)
            go to step 2.
        otherwise (print no item to delete).
Step 3: top = top - 1
Step 2: stack[top] = stack[top - 1]
Step 4: exit.
```

pop(stack, Top, &item) or (top < 0)

step 1: if (top == -1)

print underflow.

otherwise go to step 2.

step 2: &item = stack [Top]

step 3 : Top = Top-1

step 4: print deleted item

step 5 : exit.

# Queue.

## Definition

linear list of elements in which deletion can take place only at one end, called the front, and insertion can take place only at the other end, called the rear.

## Two basic operation

Enqueue
→ insert at the rear end (back end).

Dequeue
→ delete from the front end.

- It follows first in first out methodology.

whenever deletion happens

front = front + 1

whenever delet addition happens

Rear = Rear + 1

is empty( ).

To check whether the queue is empty or not.

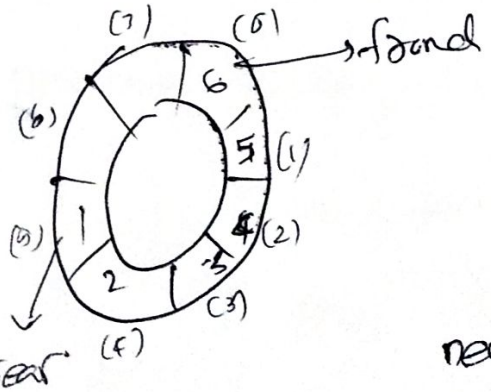
```
bool isempty()
{
 if (front<0 || front > rear)
    return true;
    else
     return false;
 }
```

22-9-2020

Even though front has vacant space we don't have insertion doesn't take place.

That's how circular queue arise

circular queue is a linear data structure.

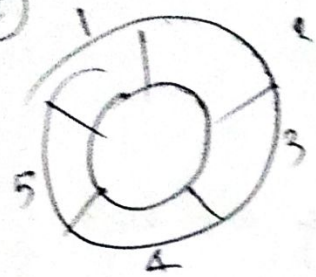



Rear = 5 and front = 0.

(6 and 5 are deleted)

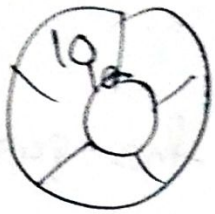
new elements are added to that position.

(1)



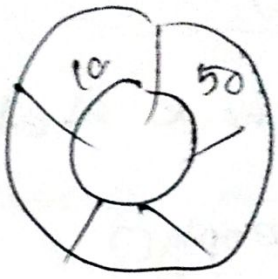

r=0
f=0

(2)




r=1
f=1

(3)




r=2
f=1


r=3

f=?

when deletion happens (from I$^{st}$ position)

front increments.

front = rear + 1, so Queue overflow

Insertion algorithm

29-9-20

# priority Queues

- Regular queue - FIFO.
- In priority Queue, an item with highest priority comes out 1st
- Here FIFO pattern is no longer valid.
- The item with higher priority must be removed befor the item with the lower priority.
- If two elements have equal priority, then they are served according to their order in queue.

Enqueue : insertion based on priority.

Dequeue : Removes the item with the highest priority.

peek :

priority Queue.

```
If (rear = maxsize-1)
        print("Queue overflow") and return
        end of if
If front = -1  and  rear = -1   ⟶ (No element in Queue)
        Set front = rear = 0
        Queue[rear] = item
        priority[rear] = p
```

else.

```
for(i=rear ;i>=0 ;i--)
if (p > priority[i]
  {
    priority [i+1]= priority [i];
    Queue [i+1] = Queu[i];
  else
   Break;
   rear = rear+1;
   priority[i+1]=p;
   Queue[i+1] = item.
```



Dequeue

1: if (front = -1 or front > rear)
   print "underflow" and return

2: Else
   item = queue [front];
   p = priority [front];
   front = front + 1
   Return item

Deque :

Types of Deque.

Input restricted deque.

- Inserted at one end
- Remove from both end.

output

- element remove from one end
- insert from both end.

Operations in deque

- Insert element at back.
- " " front.
- Remove " " front
- " " at back.

. Deque can be used as stack. ie insertion and deletion from one side.

. insert at rear, delete from front

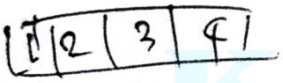
⇒ Act as queue

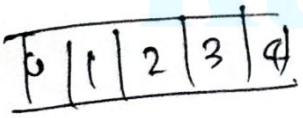
* ie LIFO and FIFO are possible in double ended queue.

So Deque can be used for the application of stack and queue.

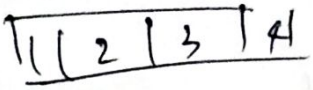
Insert front.

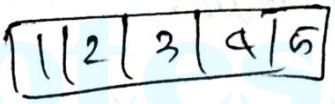
front reduces




Insert back




Remove front.

underflow (empty)

element deletes

increment front

Remove back.

rear underflow

rear — minus

## Insertion at rear (insert back)

1: check for overflow.

```
if (rear == maximum size)
    print ("Queue is overflow");
    return;
```

2. Insert element.

```
rear = rear + 1
q[rear] = new no;
```

3. Return

## Insertion at front.

1: check for front position

```
if (front <= 0)
    print (cannot add item)
    return;
```

2: Insert element.

```
else, front = front - 1
    q[front] = new no;
```

3: Return;

## Remove from front end

1. To check the Queue have element or not.

   If front=0

   print " underflow (No element)"

   return ;

2. else,

   n=q[front];

   print "n is deleted");

   front=front+1;

3. Return



## Remove from rear end

1. To check element is there.

   If rear=0

   print "underflow (No element")

2. else.

   no: q[rear];

   print "n is deleted")

   rear = rear - 1 ;

3: Return ;

# Transformation of infix to postfix expression

expression -

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


postfix string ——→ this is (add to ps, add to ps, add to ps).

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

- digit scanned is 4

  Add 4 to the postfix string

- operator scanned is $

  push $ to stack.

- digit scanned is 2

  Add 2 to ps

- operator scanned is *

  poping $ from stack. As hierarchy of $ is greater than * adding $ to postfix string.

  push * on stack.

- digit scanned is 3

  add 3 to ps

• operator scanned is -

popping * from stack

hierachy of * > that of -

∴ * ⟶ ps.

- to stack

• digit scanned ⟶ 3

• 3 ⟶ ps

- scan +

- popping from stack.

H(-) > H(+)

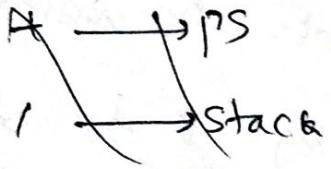
- ⟶ sp

+ ⟶ stack

• scan ⟶ 8

8 ⟶ ps

• scan /





stack

As hierachy of + is not greater than / push '+' on the stack and '/' also

- scan 4

  4 → ps

- scan /

  as H(/) = H(/)

  poping the old / from stack and then to ps

  new one to stack


- Scan (

  → To stack


- scan 1

  1 → ps.



- scan +

  H(C) ≯ H(+)

  To stack


- scan )

  → ps

- scan )

  Then + → ps

  Then ( → This encounters ')' (so cancels)

  then / → ps

  then + → ps.

③

$, power > *, %, / > +, - > (, ).

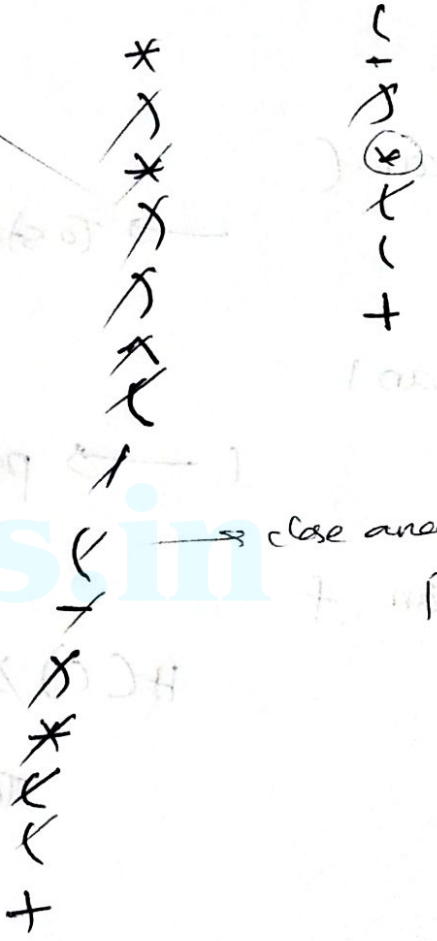
Q. A + ( ( B * C ) - ( D / ( E ^ F ) ) * G ) * H convert this to postfix.

~~A B *~~

ans: ~~A B C * - D + / E ( E ^ ) G * H )~~

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

=.




case another opere.

stack.

stackil allath, higher another compared to the coming one.

entil ⇒.

stackil allath is poped up already

Head.

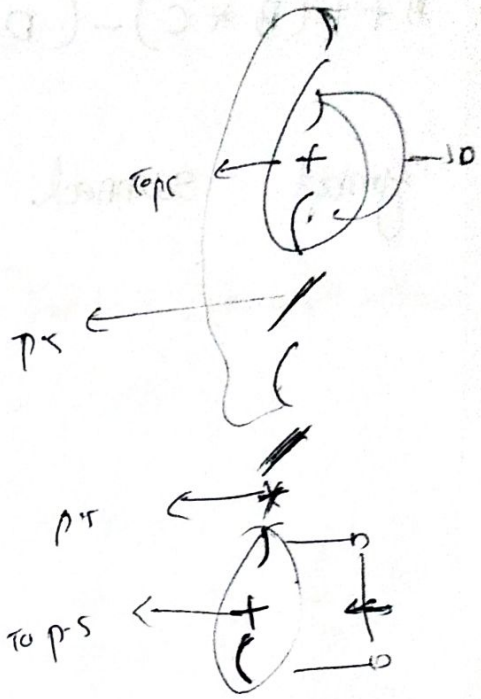
Q. A + ((B ↑ C) − (D / (E ^ F)) * G) * H.

| symbol scanned | stack | postfix expression | Description |
|---|---|---|---|

G. $(A+B) * C / (D / (J+D))$

= ~~AB+C*D~~

AB+C*DJD+//



⇒ Infix → prefix.

1. Reverse the infix expression.
2. make every ')' as '(' and every '(' as ')'
3. convert expression → postfix form
4. Reverse the expression.

Q. A+((B*C)-(D/(E^F))*G)*H.

⇒ ~~H*(G*~~

1. H * )G* ))F^E(/DC-)C*B((+A
2. H * (G* ((F^E)/D)-(C*B))+A
3. ~~H*G~~

   H * G*F E^D/C B * ~~A~~ * A +
4. ~~+A * - *BC/DAEF*G*H~~

   +A * - *BC*/D^EFGH.

- 934*8+4/-

| character | Stack. |
|---|---|
| 9 | 9 |
| 3 | 9,3 |
| 4 | 9,3,4 |
| * | 9,3,4 * |
| | = 9,12 |
| 8 | 9,12,8 |
| + | 9,12,8,+ |
| | = 9,20 |
| 4 | 9,20,4 |
| / | 9,20,4,/ |
| | = 9,5 |
| - | 9,5,- |
| | = 4 |

prefix.

++-927*8/4 12

| character | stack. |
|---|---|
| 12 | 12 |
| 4 | 12,4 |
| / | 12,4,/ |
| | =3 |
| 8 | 3,8 |
| * | 3,8,* |
| | =24 |
| 7 | 24,7 |
| 2 | 24,7,2 |
| 9 | 24,7,2,9 |
| - | 24,7,2,9- |
| | =24,7,-7 |
| + | =24,7,-7,+ |
| | =24,0 |
| + | =24,0,+ |
| | =24 |

Binary search.

- compare the middle element

If yes print that

If no find which portion contains that no.

left / right

go to left / right

then it's middle (find)

repeat until the element appears

MODULE - III

# Linked Lists

- A list refers to a set of items organised sequentially.
- Linked list are not continous memory.
- Array problems,
  - has to be specified at beginning
  - deleting, insertion may require shifting of elements in the array.



Linked list

- make each item in the list part of a structure.
- structure also contains a pointer or link to the structure containing the next item

when we use array, elements are stored in continous memory location so by increment or decrement we get the next element.

Linked list are not of continious memory location. It has a data part and next(address) part together in a node. address is provided after each data part

and using that address, next element is taken to that address, and the element can be call from this address. Here the last address is a null one.

## Self Referential structures

structure pointing to the same type of structures.

- each structure of list called node, consist of two fields

```
data item
address of next..
```

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

- each data item element must have two parts Data part and another is link (pointer) part.

# ASSIGNMENT -

1. Evaluate the following expressions written in reverse polish notation. Assume single digit operands and ^ represents exponentiation operator i) 123 * + 42 / ^ ii) 63 / 45 - *

- i) 123 * + 42 / ^

| character scanned. | stack. |
|---|---|
| 1 | 1 |
| 2 | 1, 2 |
| 3 | 1, 2, 3 |
| * | 1, 2, 3 * |
| | = 1, 6 |
| + | 1, 6, + |
| | = 7 |
| 4 | 7, 4 |
| 2 | 7, 4, 2 |
| / | 7, 4, 2, / |
| | = 7, 2 |
| ^ | = 7, 2, 1 |
| | = 49 |

prefix ↔ polish notation.

* 63/45 - *

| character scanned | stack. |
|---|---|
| 6 | 6 |
| 3 | 6 3 |
| / | 6 3 /<br>= 2. |
| 4 | 2 4 |
| 5 | 2 4 5 |
| - | 2 4 5 -<br>= 2 -1 |
| * | 2 -1 * |
| | -2 |

2. Convert the following expression into its corresponding postfix form using the prescribed algorithm. (300+23)*(43-21)/(84+7) Do the evaluation of resultant postfix expression.

~~character~~ scanned stack postfix ~~string~~.

Q. $(300+23)*[43-21]/(84+7)$

| character scanned | stack | postfix |
|---|---|---|
| ( | ( | |
| 300 | ( | 300 |
| + | ( + | 300 |
| 23 | ( + | 300 23 |
| ) | (+) | 300 23 + |
| * | * | 300 23 + |
| ( | *( | 300 23 + |
| 43 | *( | 300 23 + 43 |
| − | *(− | 300 23 + 43 |
| 21 | *(− | 300 23 + 43 21 |
| ) | *(−) | 300 23 + 43 21 − |
| / | */ | 300 23 + 43 21 − |
| ( | */( | 300 23 + 43 21 − |
| 84 | */( | 300 23 + 43 21 − 84 |
| + | */(+ | 300 23 + 43 21 − 84 |
| 7 | */(+ | 300 23 + 43 21 − 84 7 |
| ) | */(+) | 300 23 + 43 21 − 84 7 + |

∴ postfix expression

= 300 23 + 43 21 − 84 7 + / *

Evaluation.

300 23 + 43 21 - 84 7 * / *

| character scanned | stack. |
|---|---|
| 300 | 300 |
| 23 | 300 23 |
| + | 300 23 + ⇒ 323 |
| 43 | 323 43 |
| 21 | 323 43 21 |
| - | 323 43 21 - ⇒ 323 22 |
| 84 | 323 22 84 |
| 7 | 323 22 84 7 |
| + | 323 22 84 7 + ⇒ 323 22 91 |
| / | 323 22 91 / ⇒ ~~.241~~ 323 .241 |
| * | 323 .241 * ⇒ 77.843. |

answer = 77.843

7-10-2020

1. Node Declaration →

```
{
    int data;          ——— variable data part
    struct node *next; ——→ variable for pointer part
}                          (pointer that will point next node).
```

2. Declare variable

```
struct node *start   ——→ pointer to 1st node.
```

3. Allocate memory for new node.

```
start = (struct node*) malloc(sizeof(struct node));
```

start → pointer.

malloc → allocated memory.

size of struct node depends upon the memory allocated.

4. Enter value

```
start->data = item
start->next = NULL
```

arrow is used bcoz, Here is a pointer.

eg: start is a pointer

∴ start → data.

Operations.

Insertion.

- Insertion at beginning
- Insertion at end
- Insertion after a specific node.

Insertion at end →

last node, pointers → new node.

next pointer of new node → NULL

12.10.20

Deleting 1st node

making the start pointer → two.

```
node *temp=start;
start = temp->next; temp.next
delete (temp);
```

Deleting an internal node.

~~root~~ make the next pointer of the node previous to the node being deleted, point to the successor node of the node to be deleted.

Delete the node using delete keyword.

C code,

## Searching

finding the required element in the list.

Algorithm →

Type of linked list →

Single linked list, circular LL, double LL.

Circular LL →

Data part and pointer part.

Insert at end (last).

↔

deleap=

11-10-2020.

19-10-2020

# CIRCULAR LINKED LIST

1) Insert a node at the beginning.

Algorithm.

1. Start.
2. Create a new node.
3. Check whether the list is empty.

(Top == null) If yes → 4 else → 5.

4. set Top = newnode.

newnode → next = Top.

5. define a node pointer temp and initialize with top

6. Move temp to next until the last node

(temp → next == Top)

7. set

newnode → next = Top

Top = newnode

temp → next = Top.

8. Stop.

Insert at the end.

1. start
2. create a newnode.
3. check empty or not

   (Top == null)  If yes → 4  No → 5

4. set Top = newnode.

   newnode → next = Top.

5. define a node pointer temp and initialize with top.
6. move the temp upto the last node.

   (temp → next == head)

7. set temp → next = newnode.

   newnode → next = head.

8. stop.

2) Delete from the beginning

1. Start
2. check empty or not

   (Top == null)

   yes → 3  no → 4

3. display, No element to delete.

4. define two node pointers temp1 , temp2 . and initialize with top.

5. check whether only one node or not

(temp1 → next == Top).

6. True then

Top = null

and delete temp1

7. false move temp1 until it reaches last

temp1 → next = head Top

8. Then

Top = temp2 → next

temp1 → next = head

9. stop. and delete temp2.

Deleting from End of the list

1. Start

2. check empty or not.

Top == null. yes → 3 No → 4

3. List is empty , display this

4. define temp1 , temp2 pointer

initialize temp1 with head.

5. To check whether only one node is there.

(temp1 → next == Top)

6. If it is true,

Top = Null

delete temp1

7. if false

temp2 = temp1

move temp1 to last node.

(temp1 → next == Top)

8. temp2 → next = top and delete temp1

9. stop.

Aryana -1K

Tutorial

Infix to postfix.

a) A + ((B * C) - (D / (E ^ F)) * G) * H.

| character scanned | stack | postfix notation. |
|---|---|---|
| A | | A |
| + | + | A |
| ( | +( | A |
| ( | +(( | A |
| B | +(( | AB |
| * | +((* | AB |
| C | +((* | ABC |
| ) | +( | ABC* |
| - | +(- | ABC* |
| ( | +(-( | ABC* |
| D | +(-( | ABC*D |
| / | +(-(/ | ABC*D |
| ( | +(-(/( | ABC*D |
| E | +(-(/( | ABC*DE |
| ^ | +(-(/(^ | ABC*DE |
| F | +(-(/(^ | ABC*DEF |
| ) | +(-(/ | ABC*DEF^ |
| ) | +(- | ABC*DEF^/ |
| * | +(-* | ABC*DEF^/ |
| G | +(-* | ABC*DEF^/G |
| ) | +* | ABC*DEF^/G*- |
| * | +* | ABC*DEF^/G*- |
| H | +* | ABC*DEF^/G*-H |

∴ A+((B*C)-(D/E^F))*G)*H

= ABC*DEF^/G*-H*+

2) (A+B)*C/(D/(J+D))

| character scanned | stack | postfix notation |
|---|---|---|
| ( | ( | |
| A | ( | A |
| + | (+ | A |
| B | (+ | AB |
| ) | | AB+ |
| * | * | AB+ |
| C | * | AB+C |
| / | */ | AB+C* |
| ( | /( | AB+C* |
| D | /( | AB+C*D |
| / | /(/ | AB+C*D |
| ( | /(/( | AB+C*D |
| J | /(/( | AB+C*DJ |
| + | /(/(+ | AB+C*DJ |
| D | /(/(+ | AB+C*DJD |
| ) | /(/ | AB+C*DJD+ |
| ) | / | AB+C*DJD+/ |

i.e.

(A+B)*C/(D/(J+D))

= AB+C*DJD+//

21-10-20

| Arrays | Linked list |
| --- | --- |
| . Designing expensive | . Dynamic size. |
| . during insertion/deletion shifted, (elements) | . No shifting |
| . efficient index | . No random access |
| . If array is full no memory loss, else wastage of mem. | . No wastage of memory. |
| . Sequence access is faster elements are in continuous memory location | . is slow, elements are not in continuous location. |



## Memory management

process carried by the os and hardware to accomodate multiple processes in main memory,

there are two basic type of memory allocation.

- Static memory allocation.
- Dynamic memory allocation.

## Static memory allocation

Declaration of variables, structures, and classes at the beginning of a class or function →

```
Simple SimArray [50];
Simple simVal;
int x;
double d;
char ch;
```

## Dynamic memory allocation

- allocate during runtime.
- 4 library functions in C defined under <stdlib.h>
  - malloc()
  - calloc()
  - free()
  - realloc()

### malloc ⟷

single large block of memory with specified size

```
int *ptr = (int *) malloc (5 * sizeof(int));
```

ptr → [ ] a block of memory.

## Calloc()

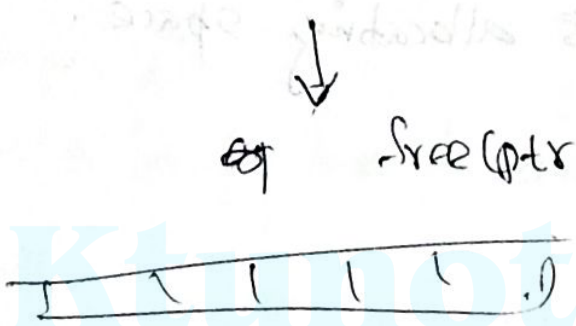
```
int *ptr = (int*) calloc ( 5, sizeof(int));
```

## free()

used to dynamically de-allocate the memory.

syntax : free(ptr).

```
int *ptr = (int *) calloc (5, size_of(int));
```



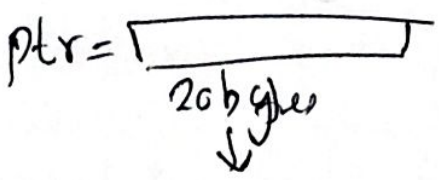
```
free(ptr
```



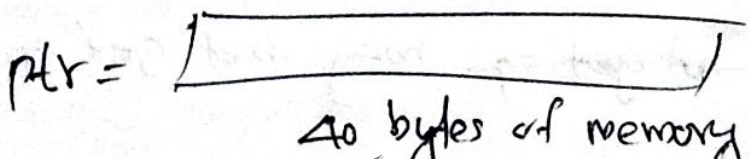
## realloc() method

used to re-allocation.

```
ptr = realloc (ptr, newsize);
```




```
ptr = realloc (ptr, 10 * size_of(int));
```



27-10-2020

different techniques are used ~~by mem~~ os for the memory allocation -

These partions maybe allocated mainly by 3 ways.

1. First-fit memory allocation
2. Best-fit " "
3. Worst-fit " "

memory allocation is allocating space.

First fit →

- Simplest algorithm.
- allocate job to the nearest memory position available with sufficient size.
- If memory required < memory block (hole).
  it is assigned to this memory.

adv.

- Fast in processing.

disadv

- waste of memory.
- so that many job may ~~not get~~ may not get space for it

Best-fit

. searches the entire list and takes the smallest hole that is adequate.

• Try to find a hole that is close to the actual size needed, Rather than breaking a hole.

adv:

• memory efficient
• No wastage of memory

Disadv

• Slow process
• takes a lot of time to complete the work.

Worst-fit

• always search the largest hole.

adv

• large internal fragmentation, so more space is there. actually is a (-ve)

Dis:

• slow process
• bcoz it traverse all the partitions.

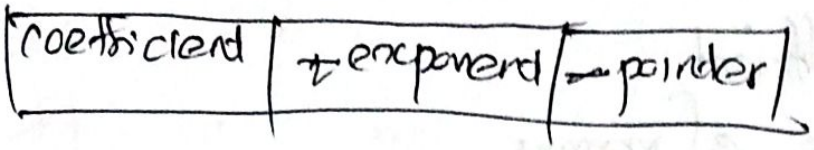
## Applications of Linked list

. can be used to implement stack, Queues.
" " " " Graphs.
polynomial representation.
memory management.

polynomial →

Waste of space.

- Data part has 2 parts.
  - coefficient
  - exponent




adv

Save space.

Easy to maintain

disadv

- can't go backward.
- can't jump to the beginning of the list from the end.

26-10-2022

## Linked list rep of Queues

1st node → front → Delete

Last " → Rear → Insert

Front = rear = null  empty

### Enqueue

Set new node as the Rear end.

Algorithm.

1. Start.
2. Declare new node ptr.
3. set ptr → Data = value.
4. if front = rear = null if yes (5) else (6).
5. set front = Rear = ptr

   set front → next = Rear → next = null
6. else.

   set Rear → next = ptr.

   set Rear = ptr.

   Rear → next = null.
7. End.

Deletion.

1: Start

2: If front = null

print underflow

3. else.

ptr = front.

set front = front → next.

4. Deleted ptr.

5. end.