Notes | Appointment time

Euclid's algorithm

- it is used to compute GCD (greatest common Divisor)
- problem
  gcd (a, b) where a > b

base part
  if gcd (a, 0) return a as gcd

inductive part
  gcd (a, b) = gcd (b, a mod b)
  where a > b

- c function for Euclid's algorithm

```
int gcd(int a, int b)
{
    int c;
    if (a < b)
        swap(a, b);
    while (b > 0)
    {
        c = a % b;
        a = b;
        b = c;
    }
    return c;
}
```

```
int gcd(int a,
        int b)
{
    if (a < b)
        swap(a,b)
    if (b == 0)
        return a;
    else
        return gcd(b, (a % b));
}
```

time complexity : 1.44 log n

time | Appointment | Notes

8 * Exponentiation

9 problem: to find x^n

10 base part: if n=0 return 1
if n=1 return x

11

12 inductive part
if n is even
then $(x^2)^{n/2}$
else $(x^2)^{n/2} * x$

13

14 time complexity: O(log n)

15 c function

```
float pow (int x, int n)
16 {
    if (x==0)
17      return 1;
    else
18  if (x==1)
        return 1;

19
    if (n%2==0)
20      return pow(x*x, n/2);
    else
21      return pow((x*x, n/2)*x);
}
22
```

chapter 2 : ADT, stack, queue and list

ADT

Abstract data type
- in which about data type and about its functionality is mentioned.
- instances is detail about data type without any programming language constraint
- while in operations, what to be done in data type is mentioned but not how to be done.

ADT = Data type + function name with behavior

ede.

Array as an ADT

AbstractDataType Array
{

Instances: it is collection of items
under same name
all items are indexed
and store in continious mem locations

Operations:
insert() to insert new element in an array
delte() to remove an element from an array

}

List ADT

AbstractDataType List
{

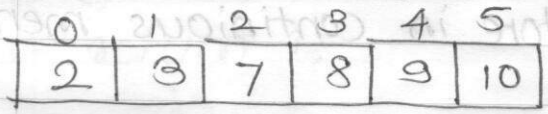
Instances: • collection of items
• in sequential order
• all elements are stored in sequential memory locations (array implementation)
• or in different memory location but knowing each other (pointer)

Operations:
insert(): insertion of an element in list
delete(): deletion of element from list
search(): searching of element in list
display(): traversing in list and displaying values of all nodes.

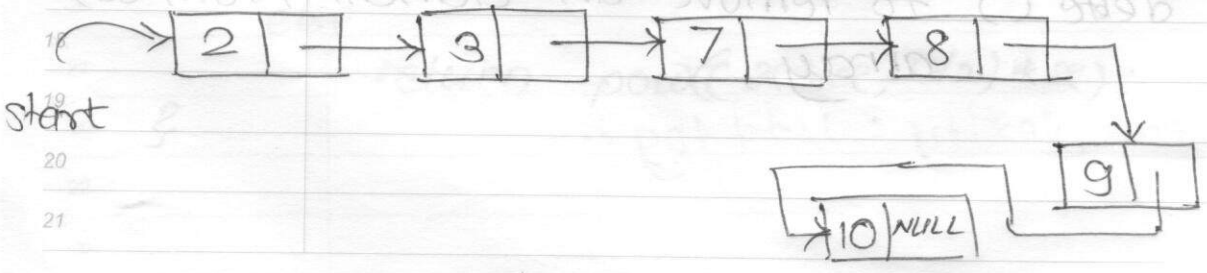
5 Sunday
}

Example
elements: 10, 2, 3, 9, 8, 7

array implementation / static list



- pointer implementation / dynamic list / linked list



Notes | Appointment time

linked list

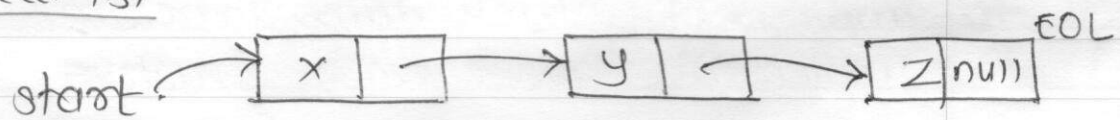
- it is made up of nodes
- where node is data structure consist of value part and address part
- value part holds a value of current node while address part holds address of next node (if any).
  If there is no node ahead of any node then such node contains address part as null. It indicates end of list (EOL)
- there is a special pointer associated with each list called as start pointer which holds address of first node in linked list.
- if start=null then linked list is empty.

ex.

I) empty linked list

start → null

II) non-empty linked list



time Appointment Notes

linked list V/s Array implementation of list

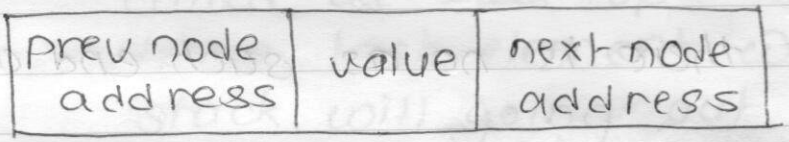
linked list	Array
1) linked list is collection of nodes, every node contain data & address (of next node)	collection of similar data type elements in indexed manner
2) access of random element is in sequential access only	can jump to any element directly
3) physical deletion of data	logical deletion of data
4) insertion & deletion of data is easy	insertion & deletion is difficult.
5) memory allocation is dynamic, not predefined	memory allocation is static, predefined.

Notes | Appointment time

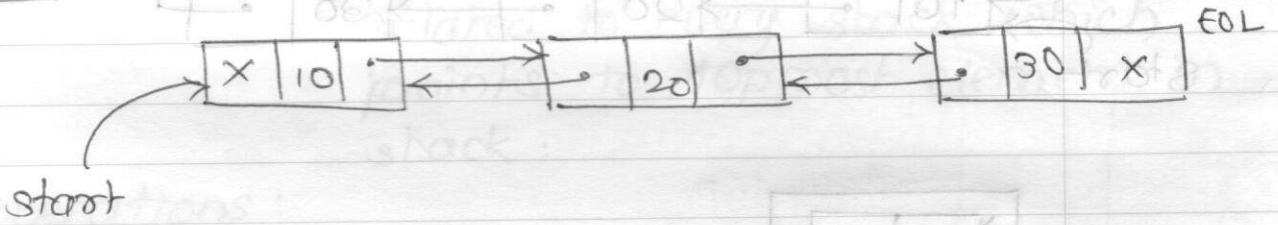
doubly / double linked list

it is a linked list where each node contains three part

1) data part : holds the value.
2) prev addr: holds addr of prev node
3) next addr: holds the address of next node.



eg.

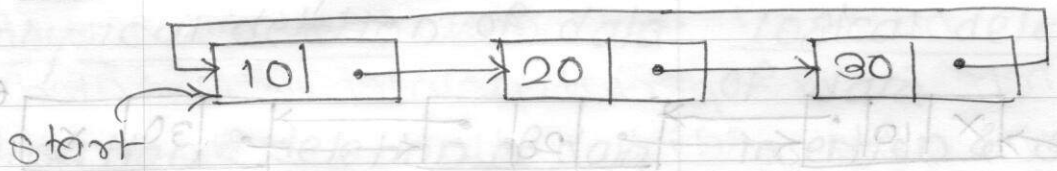


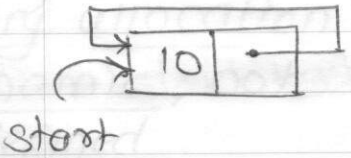
x: null

time Appointment Notes

circular linked list

- it is a linked list where each node contains 2 parts (value/data part and address part).
- all nodes will contain the address of next node in list
- but last node contains address of first node
- Hence there is no as such end of list.
- ex.





Applications of linked list

- to evaluate polynomial equation
- for handling set operations
- to implement stack data structure
- to implement queue data structure

Notes | Appointment time

Stack

AbstractDataType Stack
{

instances: • stack is collection of elements
• where insertion & deletion is performed from only one end called as top.
• hence last element which is in stack will going out first.
• stack top is special pointer related to every stack which points to topmost element in stack.

operations:

stack_empty() : if stack top is null then stack is empty (stack contains nothing)

stack_full : if stack top is equal to size of stack then stack is full

push : inserting new element on top of stack top iff stack is not full

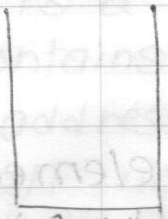
pop : removing stack top element iff stack is not empty

}

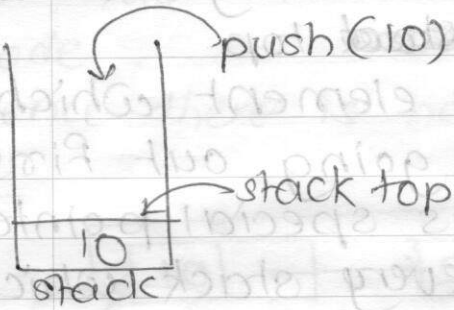
time | Appointment | Notes

ex. elements to insert: 10, 20, ~~[illegible]~~
delete

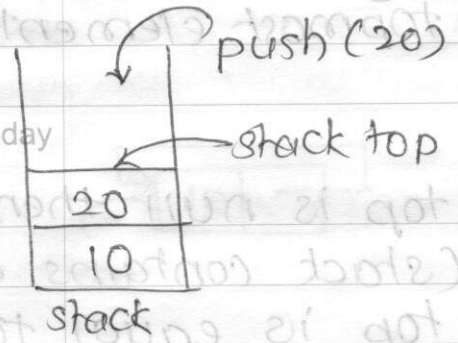
I)



II)

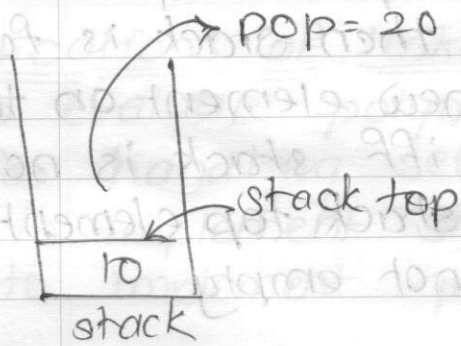


III)

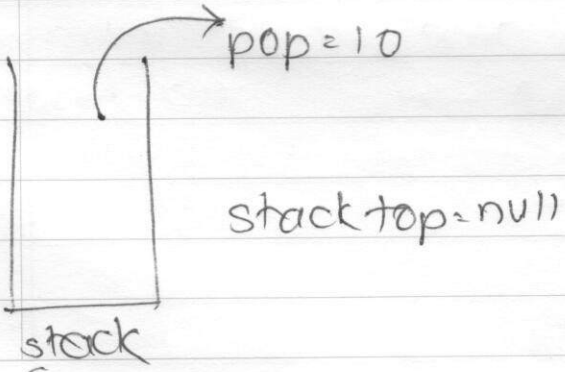


12 Sunday

IV)



V)



Notes | Appointment time

Applications of stack

1) expression conversion
2) expression evaluation
3) parenthesis parsing
4) storing function call
5) reversing a string

Expression

every expression is made up of operators and operands
there are 3 types of expression

1) infixe expression
   format: operand, operator operand$_2$
   ex: a+b
   use: natural way of representing an expression understandable for humans.

2) prefixe expression
   format: operator operand, operand$_2$
   exe: +ab
   use: used where loading processor dedicated for operation (making it ready) is more time consuming than loading operands from memory.

3) postfixe expression
   format: operand, operand$_2$ operator
   ex: ab+
   use: used where loading / fetching parameters to perform operation is time

time Appointment Notes

Algorithm to convert infix to postfix

assume algorithm knows following operators along with their priority

operator	priority level
^	4
/	3
*	2
+ -	1
( )	0
• (€OE)	–

1) take an infix expression
2) append • (end of expression €OE) at end of infix
3) read infix expression from left to right, symbol by symbol
   basis on symbol read perform some operation, by modifying / not modifying stack.
   pushing a low priority operator on top of high priority operator is not allowed (except paranthesis)
   In such case pop all high priority operator from stack and write in postfix expression.
4)

	symbol	operation
①	^	push (^)
②	/	pop until stack top = ^ or stack top = / push (/)
③	*	pop until stack top = ^ or / push (*)
④	- +	pop until stack top = ^, / or * push (+) or push (-)
⑤	(	push (()
⑥	)	pop and write in postfix until stack top != (pop (but dont write in postfix
⑦	.	pop all & write in postfix
⑧	operand	write in postfix

Ex. (1)

infix : a+b.

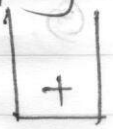
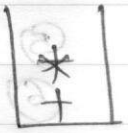
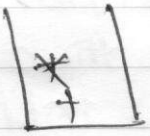
postfix : ab+

	symbol	operation	stack
1)	a	write (a)	empty
2)	+	push (+)	+
3)	b	write (b)	+
4)	.	pop & write all	empty

Time | Appointment | Notes

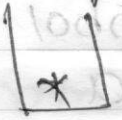
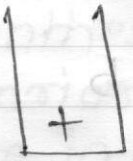
(II) Infixe : a+b*c.

postfixe: abc*+

	symbol	operation	stack
1)	a	write(a)	empty
2)	+	push(+)	
3)	b	write(b)	
4)	*	push(*)	
5)	c	write(c)	
6)	.	pop & write all	empty

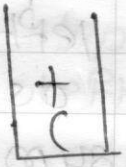
(III) Infixe: a*b+c.

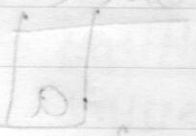
postfixe: ab*c+

	symbol	operation	stack
1)	a	write(a)	empty
2)	*	push(*)	
3)	b	write(b)	no change
4)	+	pop * write(*) push(+)	
5)	c	write(c)	no change
6)	.	pop & write all	empty

Notes | Appointment | time

(IV) infix: (a+b)*c.
postfix: ab+c*

	symbol	operation	stack
1)	(	push (( )	
2)	a	write (a)	no change
3)	+	push (+)	
4)	b	write (b)	no change
5)	)	pop +	empty
		write +	
		pop (	
6)	*	push (*)	
7)	c	write (c)	no change
8)	.	pop & write all	empty




time Appointment Notes

Convert infix expression to prefix expression

Algorithm:

1) take infix expression
2) convert to postfix expression
3) append) (end of exp) at end
4) read modified postfix expression from step 3 symbol by symbol, from left to right; perform operation as follows

symbol	operation
operand	push
operator	pop 2 operands (op_1 & op_2) write in prefix as operator op_2 op_1 push in stack
)	pop all & write

19 Sunday

infix:
ex: (a+b)*c
postfix: ab+c*
prefix : *+abc

	symbol	operation	stack
1)	a	push(a)	a
2)	b	push(b)	b, a
3)	+	pop b, a push (+ a b)	+ab
4)	c	push (c)	c, +ab
5)	*	pop c, +ab push *+abc	*

Notes | Appointment time

Evaluation of an expression

Algorithm

1) read operands & resp. values.
2) read infix expression
3) convert to postfix expression
4) append '$' at end of postfix expression
5) read postfix exp from step 4 symbol by symbol, from left to right basis on symbol perform operations as follows;

symbol	operation
operand	fetch value, & push (value)
operator	fetch 2 operands from stack (pop) as op_1, op_2 evaluate as op_2 operator op_1, push evaluated value
$	pop result &

ex. infix: a+b
postfix: ab+.
values: a=10, b=20.

symbol	operation	stack
a	push (10)	10
b	push (20)	20 / 10
+	pop 20, 10 push (10+20=30)	30
.	pop 30 as result	

time Appointment Notes

Queue

AbstractDataType queue

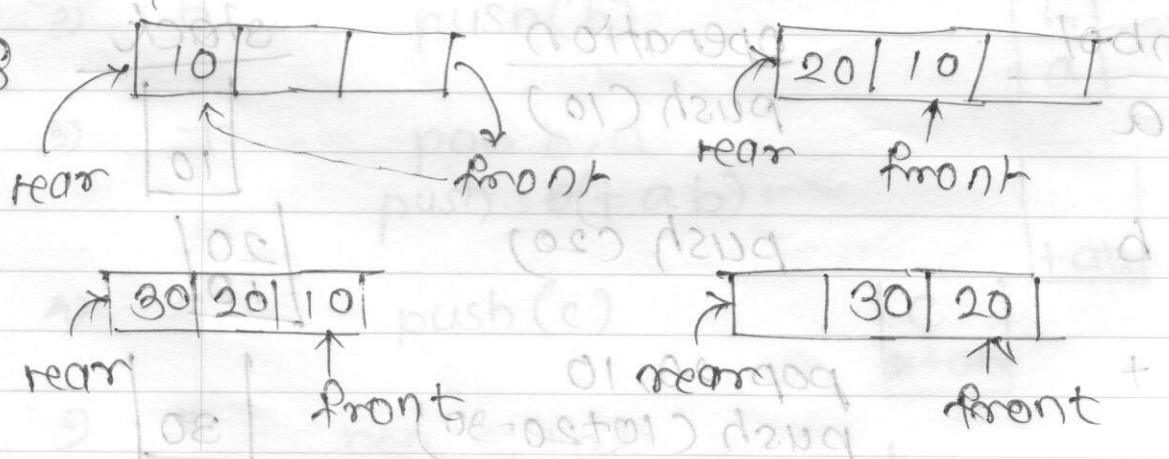
{

instances: • collection of elements
• in which element can be inserted from one end (called rear)
• and element can be deleted from another end (called front)
• hence elements are stored in FIFO (first in first out) manner.

Operations:
empty() - to find queue is empty or not
insert() - insert new element in queue from rear end
delete() - if queue is not empty then remove an element from queue's front end.

}

ex. insert 10, 20, 30



Notes | Appointment | time

Applications of Queue

i) printer queue to maintain printing jobs

ii) job scheduling algorithm (for FIFO processor) where process which comes first will be served first by processor.

iii) used in simulation and modelling

~~iv)~~