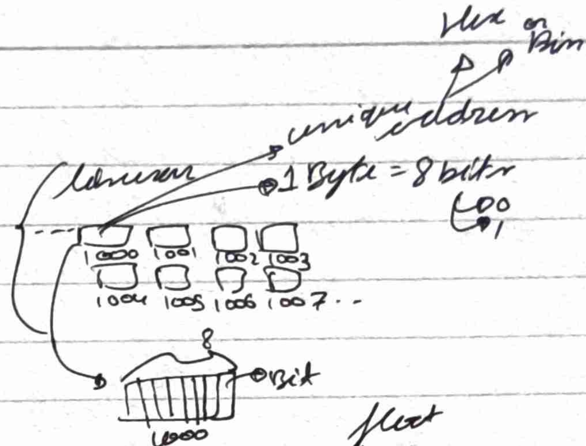
etn

data structures and algo.

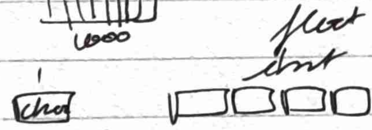
pointers

memory

pointer — Ref. memory address (3.14)



Address of variable



Var. add + N = base address + (N * (size of datatype))   4   8 double

int var = 10;

printf("%d", &var); → 1024

" " " var+1" → 1024 + 1 ✗

→ 1024 + 4 => 1028

var+5 → 1024 + 20 => 1048

Application of pointers

→ embedded system pointer linked to a device for I/O

→ memory efficient

copy of some stuff { var, func(var) ✗

memory efficient { var, func(*var) ✓

→ Dynamic Memory Allocation

↳ create, Resize, delete } at Runtime

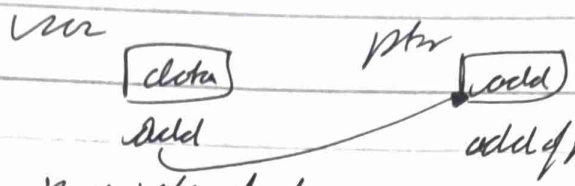
→ syntax :- datatype *ptr_name;

int *ptr;

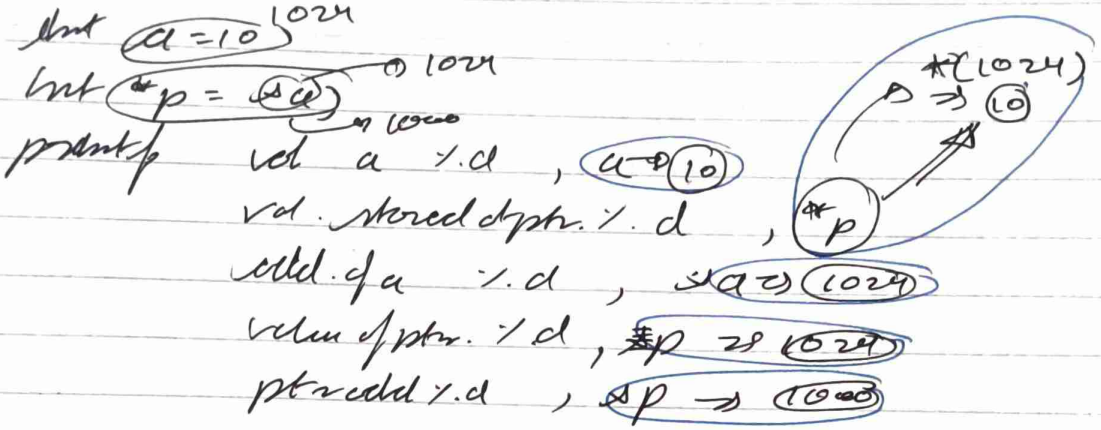
int *ptr1;

double *ptr2;

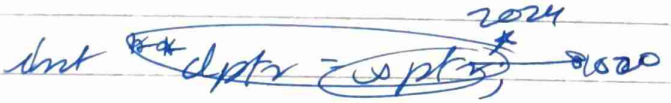
pointer



* p → value of ptr.
p → value of (ptr. which is i-e. address it contains)
& a → address of a



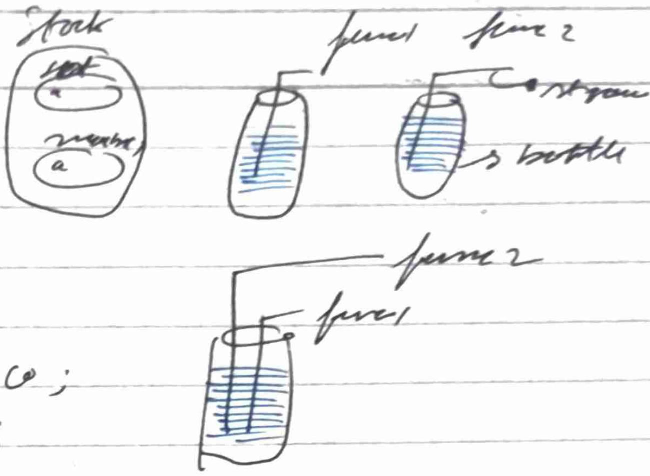
pointer to pointer
double ptr.



~~add~~ address print → ⁄. p
value " → ⁄. d

* ptr = *(1024) = 10
** dptr = *(*1024)
= *(1024)
=

call by value   int a = 10;
set(a);
→ this a is not a outside it.

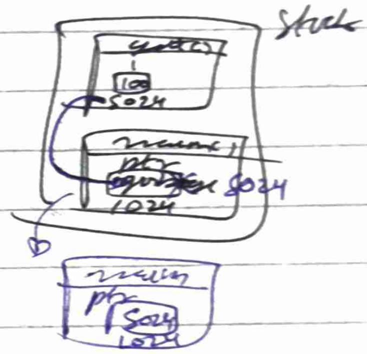


call by reference   int a = 10;
set(&a);

void set(int *a)

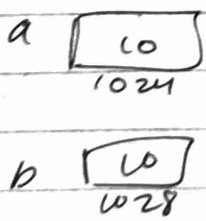
func. returning pointer   returntype* func(parameter);

int* get { int i = 100; return &i; }
int main { int *ptr = get(); return 0; }



pass ptr. to a func.   returntype func(int*)
→ address of var. is not value of var.

void add(int* a, int* b)
{ ≡ }
→ 1024   → 1028



Pointer Arithmetic

Increment of ptr.

```
int i = 10;
int *ptr = &i;
ptr++
printf("%p", ptr);   1024

ptr++;
printf("%p", ptr);   1028
```

decrement of ptr

```
ptr--;
```

ptr addition

```
ptr = ptr + 3
```

add 3 blocks: 1024 + 12 => 1036

ptr subtraction

same

```
ptr = ptr - 3
```

1012 ptr

* char 1
* float 4
* double 8

* limitation of ptrs
  - → × ✗
  - → ÷ ✗
  - → Bitwise op ✗

pointer comparison

- ==
- !=
- <
- <=
- >
- >=

```
if (p == NULL) { — }
if (p != NULL) { — }
```

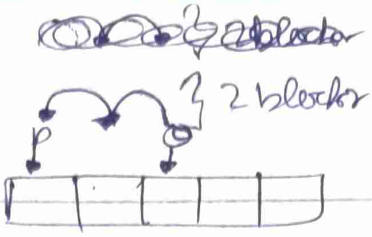
```
int arr[5] = { — };
int *p; p = &arr[0];
```

eg:-

```
int arr[5] = { — };
int *p, *q;
p = &arr[0];
q = &arr[2];
if (p < q) { printf("q ahead of p"); }
```

(1024 < 1032) → q ahead of p

* preferable when elements are pointed are of same array



↳ prev. code

1024 d

Subtracting 2 ptrs

printf("Q-P = %ld\n", Q-P); Q-P=2

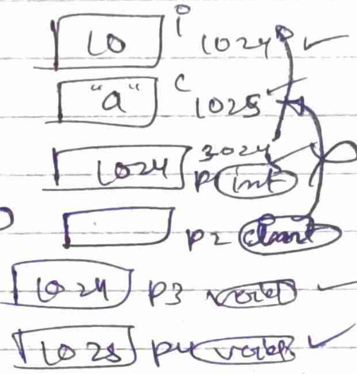
P-Q = %ld P-Q (P-2=-2) or p<Q

void pointer

↳ general purpose pointer

↳ point to any datatype var.

```
int i=10;
char c='a';
int *p1=&i;
int *p2=&c; ✗
void *p3=&i;
void *p4=&c;
```



can point to any datatype

Referencing ↳ assign ptr to var.

& ↳ reference var.

Direct access to value of var. instead of memory address

Dereferencing a normal pointer.

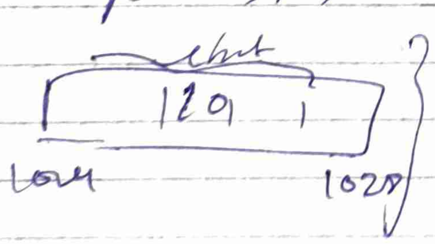
↳ * is deference opt.

printf("%d", *p3); ERROR

printf("%c", *p4);

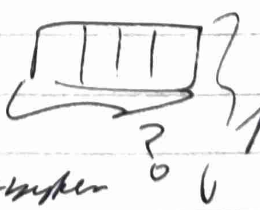
error: void ptr don't know var. datatype after Referencing.

printf("%d", *p2);



Read by compiler.

Compiler doesn't know how many bytes to Read.



Solution is Typecasting.

%d, *(int*) p3 (4 bytes)

%c, *(char*) p4 (1 byte)

p3 → p3 can read.

Application
of void pointer → Malloc

[int * malloc(int){} — int *ptr = malloc();
float * malloc(float){}] → any datatype — {void * malloc() {...}}

{int *ptr = malloc(sizeof(int));
in typecasting explicit done by compiler

Pointer arithmetic on void pointer → standard C (MSVC) → for typecast (int*)ptr+1
void → unknown size
→ GCC void size = 1 → ptr+1

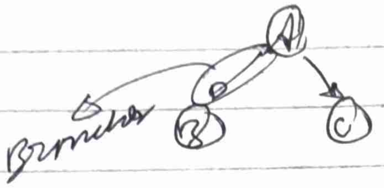
dangling ptr.
→ non existent address ptr.
"we have to free or make null ptr for safe execution process."
→ [illegible]

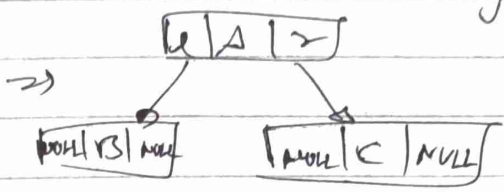
1^2 Cotola control

BTree Basics

Arr → linear

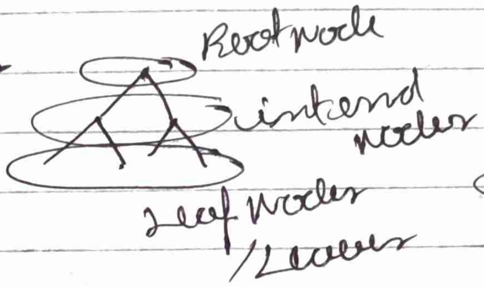
Tree → non-linear

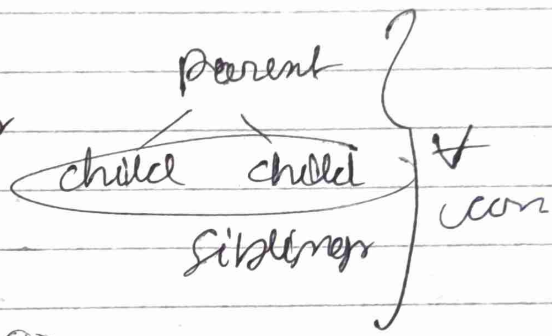


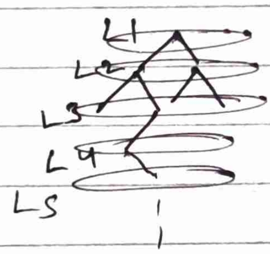


n nodes

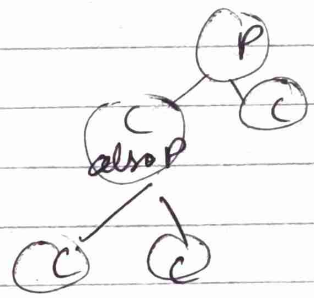
n-1 branches

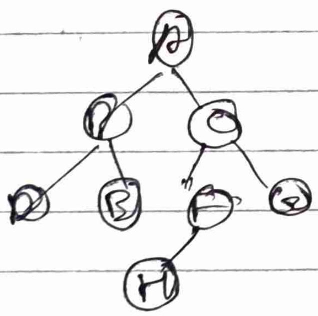






Root node ⇒ L1 only





descendents of Node are Nodes including children, grand child, great grand child etc.

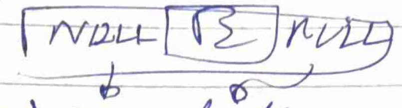
D is descendent of A, and is great grand child of A

Ancestor — — — Node which is — — — parent great grand — — etc.

A is Ancestor of D

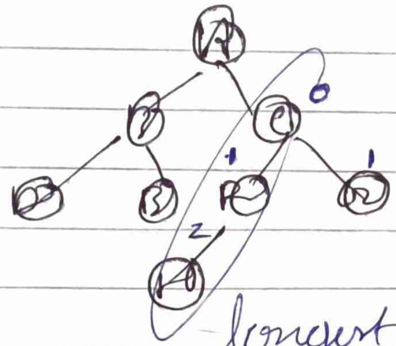
depth of the node

no of edges of node



=) 2 # depth

height of the node



ht. of c node = 2

longest c.pou

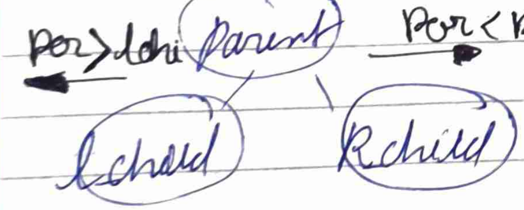
ht of root = 3

Need of BST

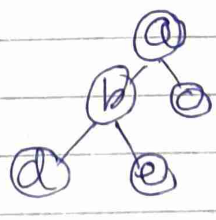
array, M
traversal, "
insert, "
delete, "

} O(n)

L> static, L> dynamic
won't grow, shrink on demand.



((D lchild < parent < R_child



Min = d
left most node

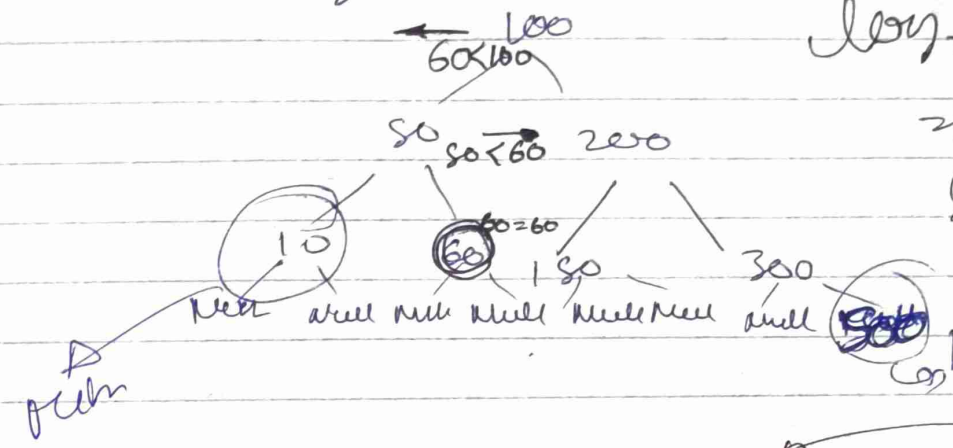
Max = c
Right most node

* duplicate not allowed in BST

element

100, 50, 200, 10, 60, 150, 300, 500 → n = 8

head



$\log_2 8$

⇒ 3 times comparison.

Only 3 steps to reach any element.

Instead of 8 elements search

n after Reduced $= n/2^0$

$n/2$

$n/4 = n/2^1$

$n/8 = n/2^3$

⋮

$1 = 1$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 2^k = \log_2 n$

$k \log_2 2 = \log_2 n$

$k = \log_2 n$

Some complexity

search & delete node from BST

find 60

find 150

100 < 150

200

150 < 200

150

150 = 150

code :-

Linear Recursion. fun(n){ return fun(n-1); }

Binary recursion int fun(n){ return fun(n-1)+fun(n-2); }