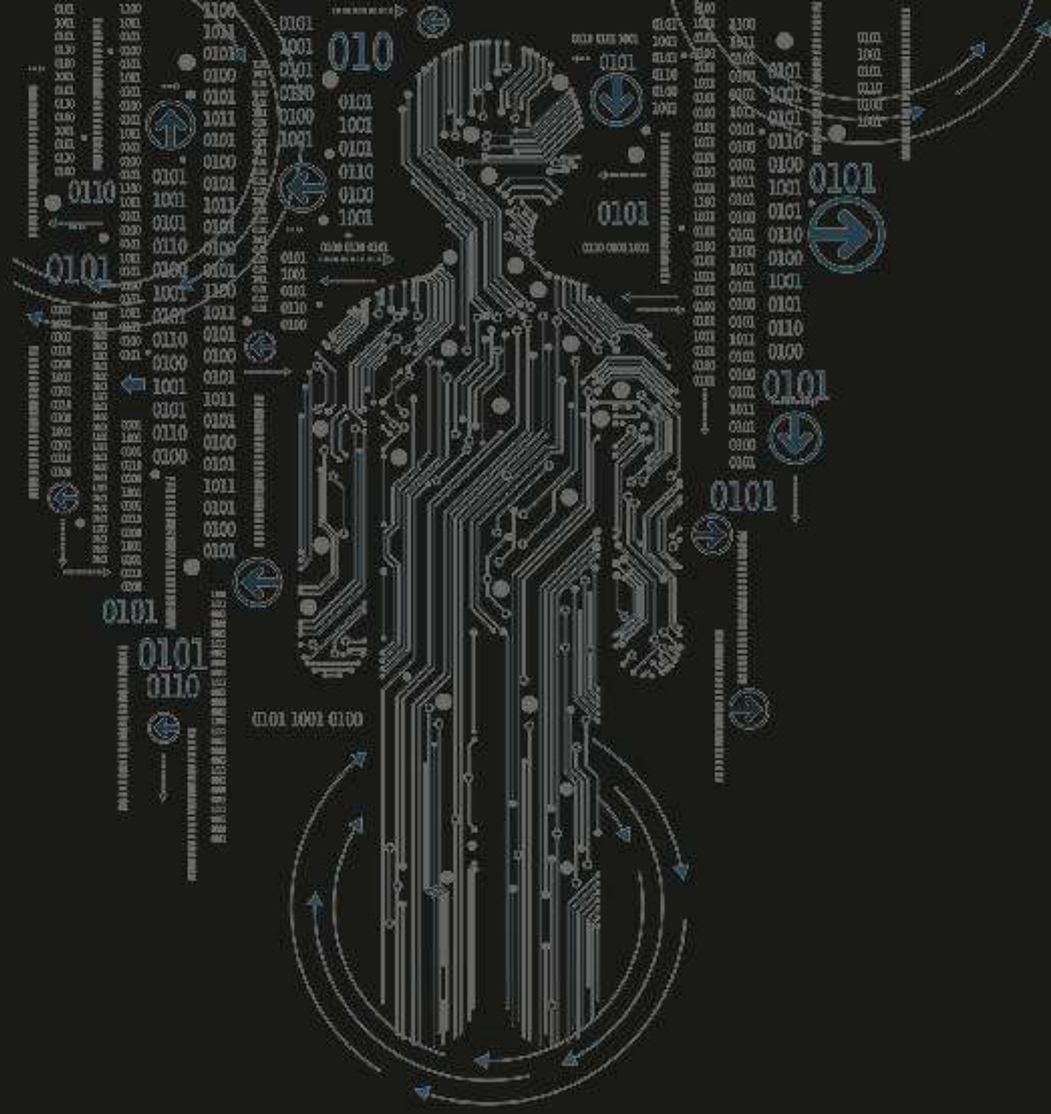
BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

C Tokens - 1



ABOUT ME

BYJU'S EXAM PREP

- Teaching Experience: 14+ Years
- Achievements: B.Tech, M.Tech, (Ph.D.)
- Achievements:
  - CS Senior Faculty
  - Mentored More Than 30,000+ Students
  - GATE Qualified
  - UGC-NET Qualified
  - SET Qualified
  - CISCO Certified

SATYA NARAYANA KAITEPALLI

BYJU'S EXAM PREP

Computer Science & Engineering

Programming and Data Structures

C DS

Programming in C, Recursion, Arrays, stacks, queues, linked lists, trees, binary search trees, binary heaps, and graphs.

Last 5 Year's Trend (2018-2022) for GATE

PROGRAMMING AND DATA STRUCTURES	
No. of Questions	5 to 6
Marks	8 to 12
Frequently Asked Topics	Recursion, Pointers, Arrays, Stack, Trees

Weightage

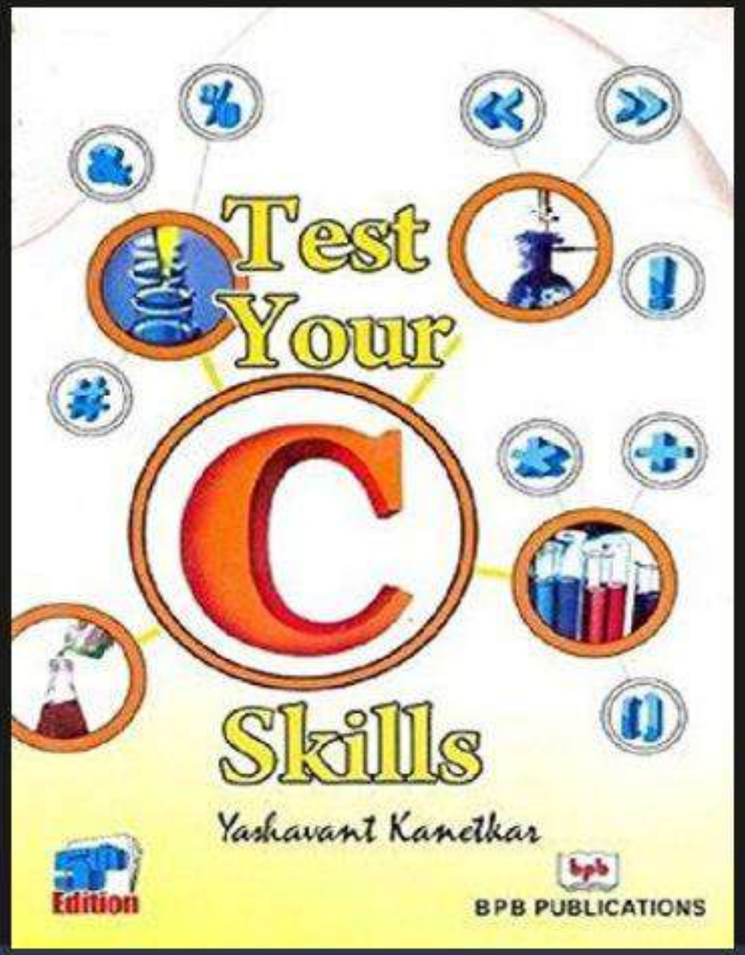
BYJU'S EXAM PREP

Programming and Data Structures

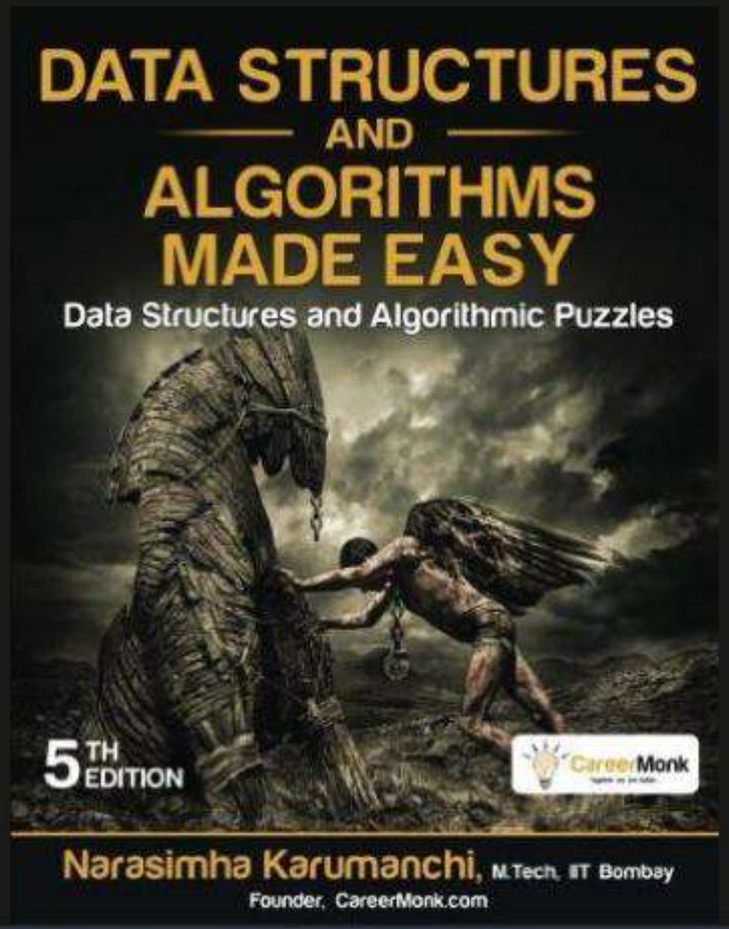
YEAR	2022	2021	2020	2019	2018	2017	2016	2015	2014	2013
MARKS	11	8	12	12	10	14	12	18	13	10

8+ Marks

Reference Books



Book Name: Test Your 'C' Skills
Author: Yashvant Kanetkar
Publisher: BPB Publications



Book Name: Data Structures & Algorithms Made Easy
Author: Narasimha Karumanchi
Publisher: Career Monk Publications

Guidelines to Attend Live Class

BYJU'S EXAM PREP

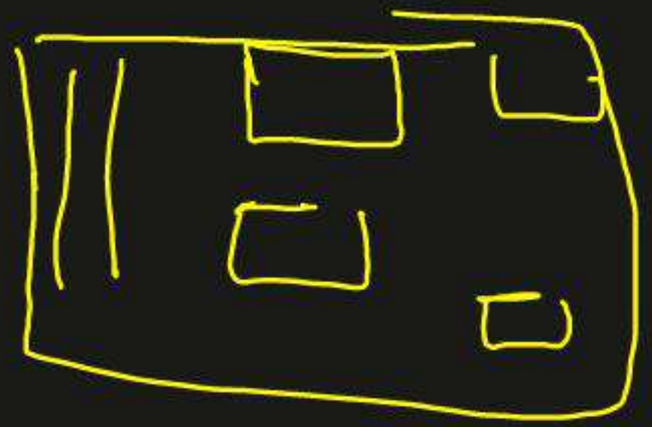
- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

CPU, Memory, I/P, O/P

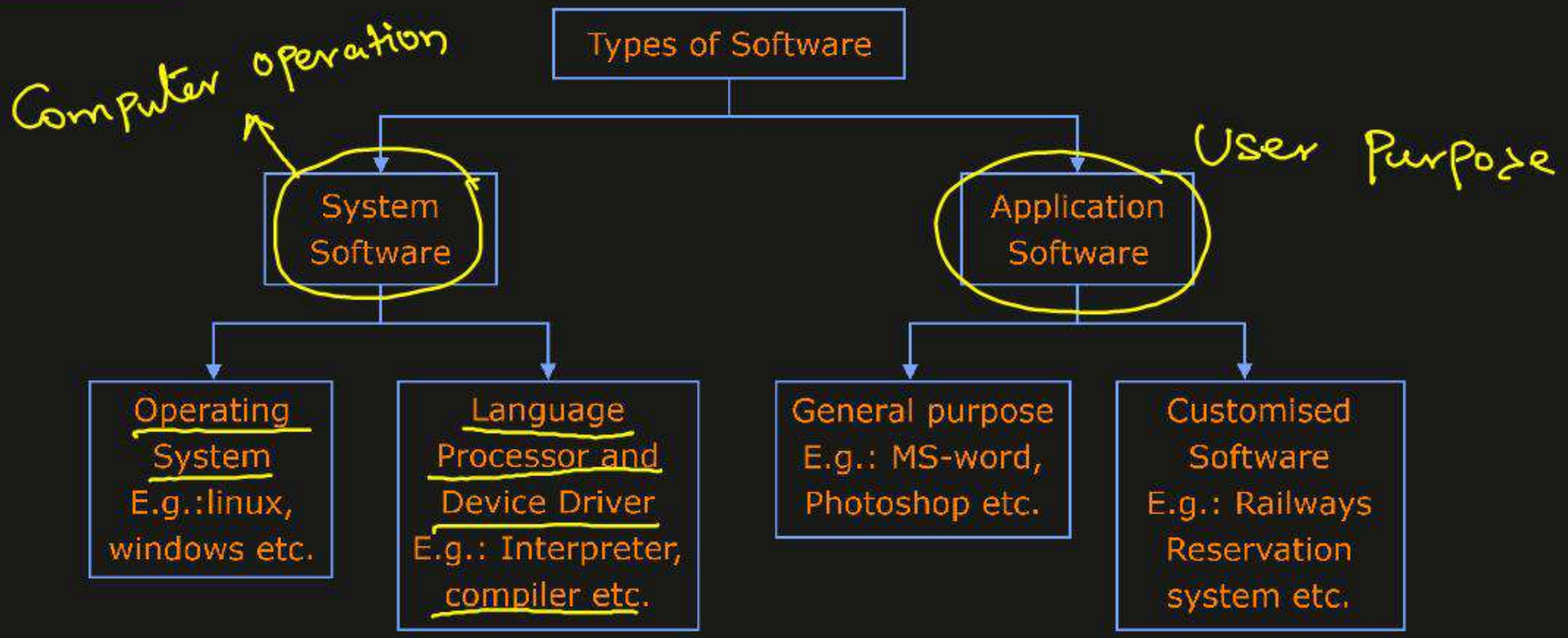
Computer ?

- It is an electronic device, which accepts input, process it, store and produces results.
- A Computer is Combination of Hardware and Software.
- Computer Hardware include CPU, Memory Devices, I/O devices and other peripherals.
- A software is set of instructions expressed using any programming language.



BYJU'S EXAM PREP

S/W, Types



Software Types

BYJU'S EXAM PREP

Programming Languages

Directly | Easily understandable by Machine.

- Computer Languages
  - Low Level Languages
    - Machine Language
      - Example - 0's and 1's
      - Binary Language
    - Assembly Language
      - Example - ADD A, SUB B, etc.
  - High Level Languages
    - Example - C, C++, Java, Phython, PHP, Pascal, etc.

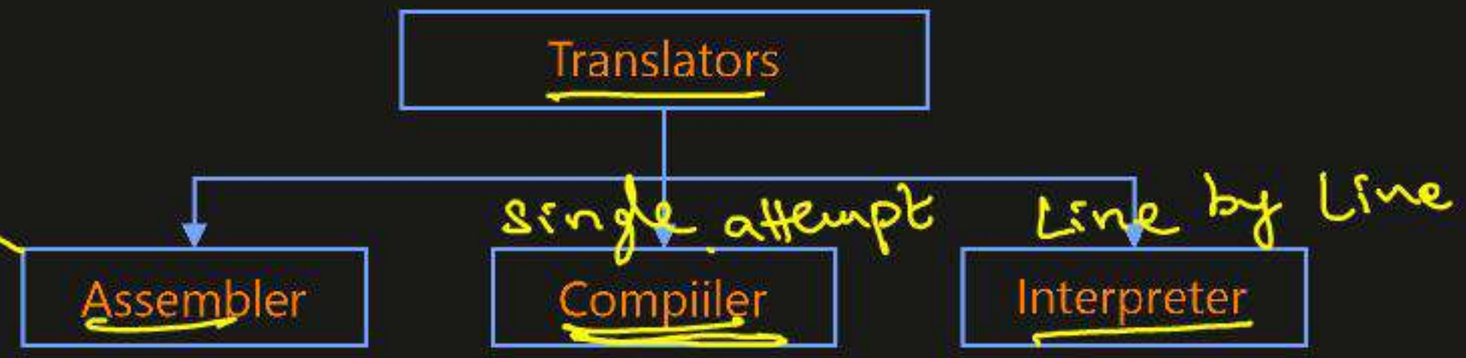
Types Of Computer Languages

BYJU'S EXAM PREP

Translators

- A translator is a program, that translates one form of language to another.
- There are 3 types of translators : Assembler, Compiler, Interpreter.

Assembly lang
– M/c lang.



Translators

HLL – M/c

ALL — M/c

BYJU'S EXAM PREP

'C' Language History

- It is written by Dennis.M.Ritchie in the year 1972
- At AT & T Bell Labs, U.S.A It is developed.
- 'C' is successor to ALGOL, BCPL and B → "C"

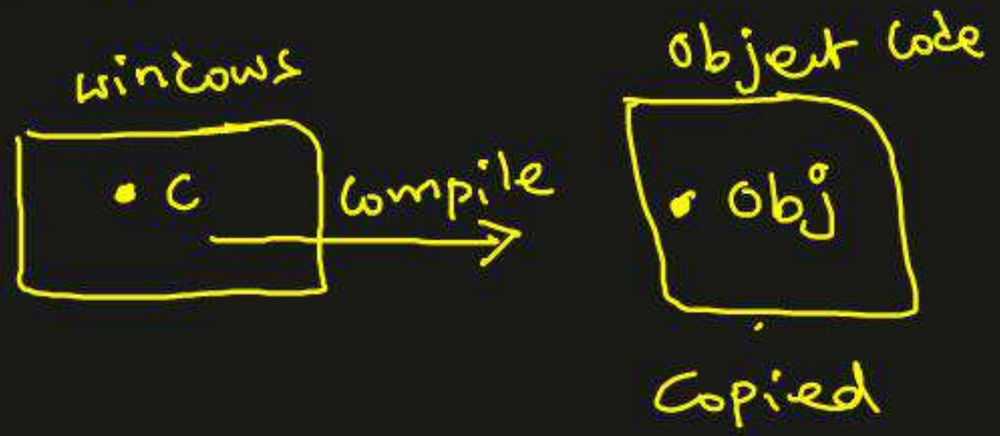
(C A)

'C' Tokens - 1

'C' Characteristics

Balaji ≠ balaji ≠ bALAJi
≠ BALAJI

- 'C' is Compiled Programming Language : compiler
- 'C' is Platform - Dependent
- 'C' is Procedure – Oriented
- 'C' is Structured Programming Language
- 'C' is Robust (C simple)
- 'C' is Portable
- 'C' is Case – Sensitive a ≠ A
- "'C' is Middle Level Language"



BYJU'S EXAM PREP

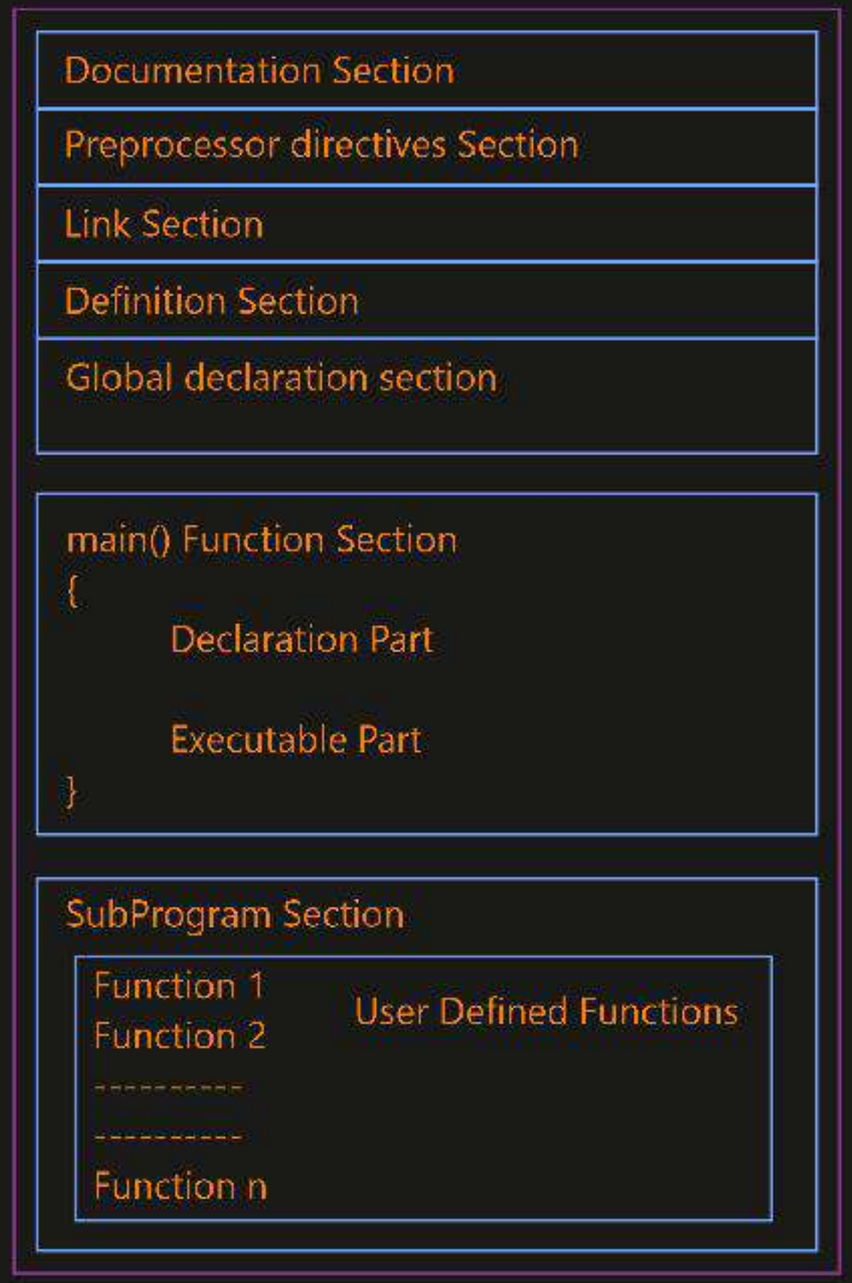
'C' Character Set

Types	Character Set
Uppercase Alphabets	A, B, C, ... Y, Z
Lowercase Alphabets	a, b, c, ... y, z
Digits	0, 1, 2, 3, ... 9
Special symbols	~ ' ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' <> , . ? /
White spaces	Single space, tab, new line.

'C' Tokens - 1

Structure of 'C' Program

To be Contd...

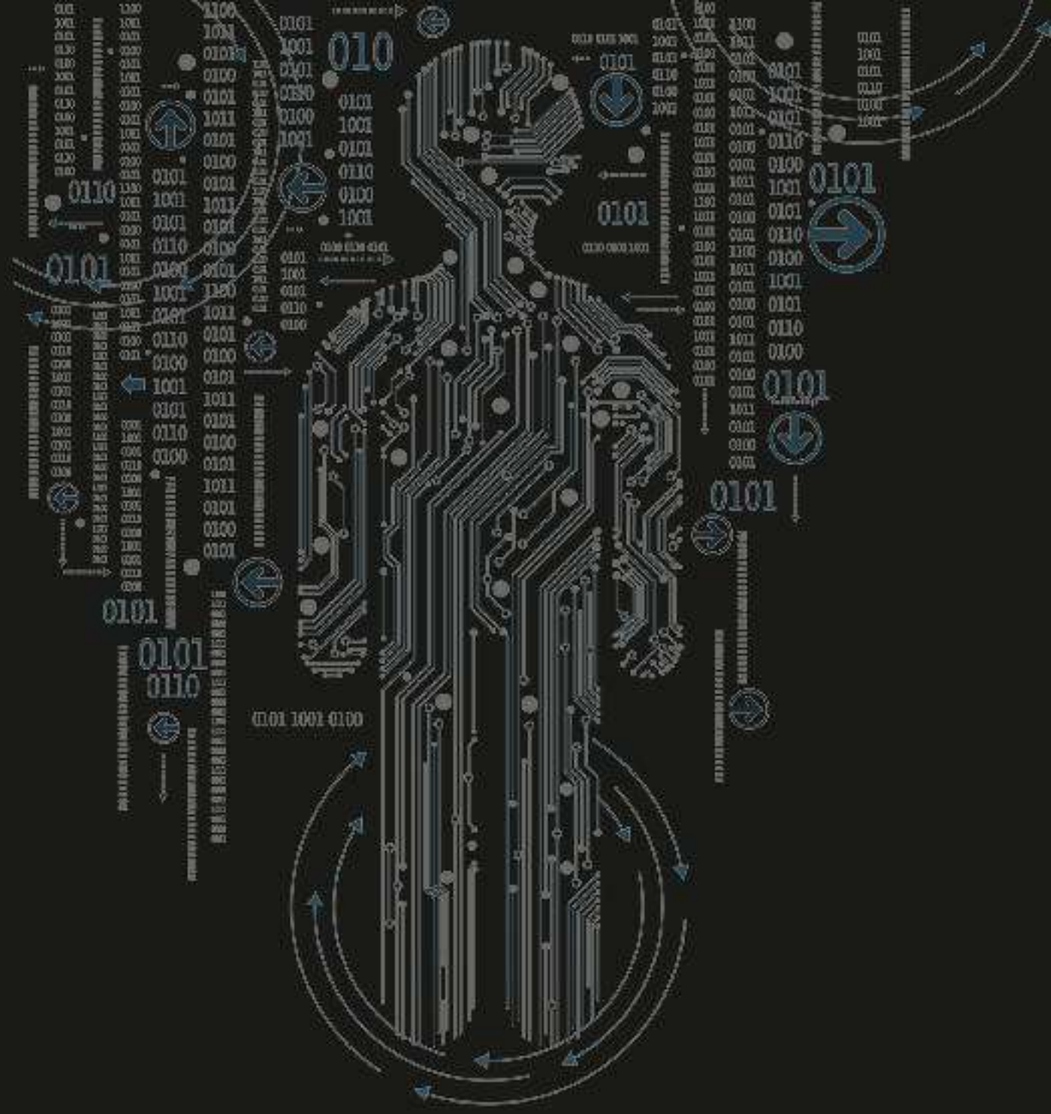


'C' Program Structure

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

C Tokens - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

C Tokens - 2

BYJU'S EXAM PREP

Last Class – Quick Revision

S/w
- System s/w
- Application s/w

Prg lang
- Low level
- High level : "C"

'C' — D. Ritchie, 1972, AT&T Bell labs

ALGOL — BCPL — B — "C"

'C' characterstics, 'C' - character set

C Tokens - 2

conio.h

#include <stdio.h>

Structure of 'C' Program

Comments - 2 types

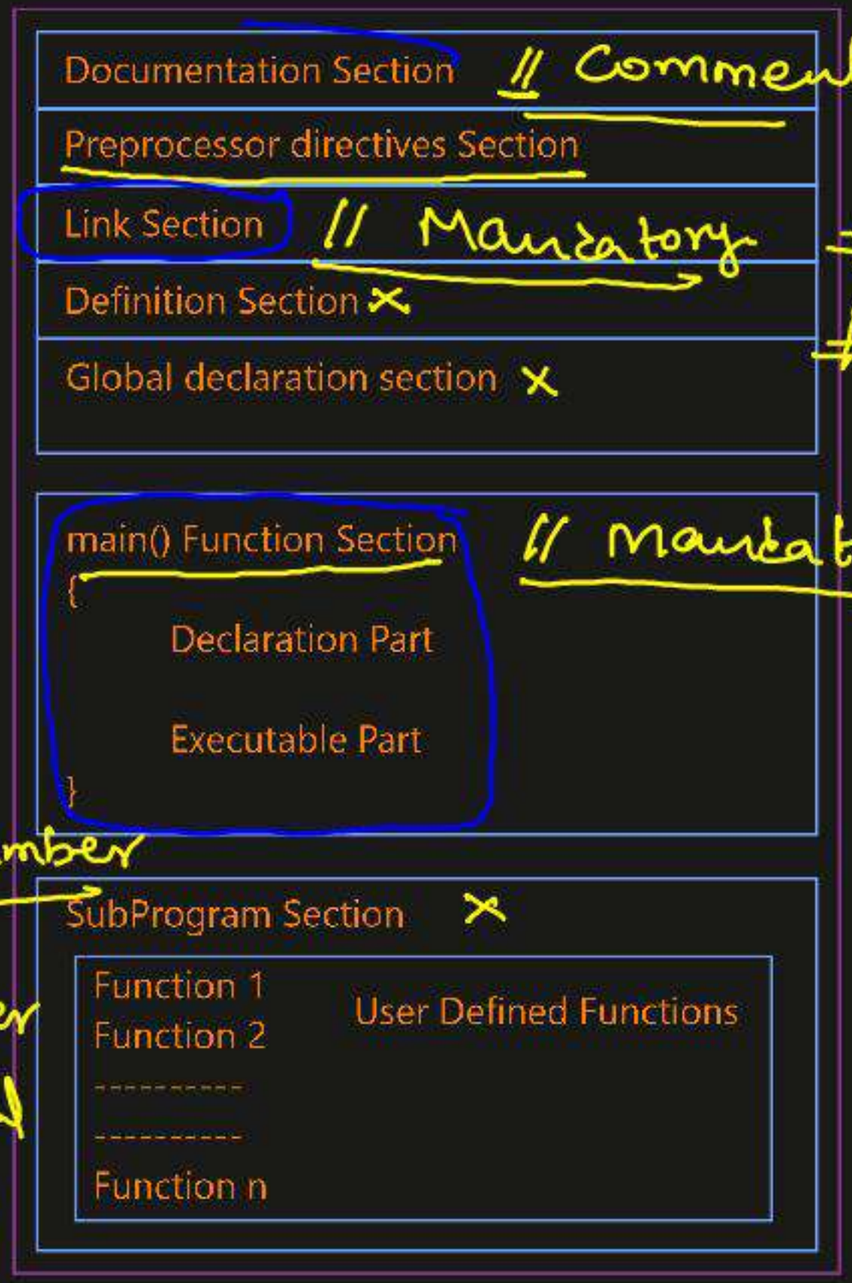
Single-line : // comment

Multi-line : /* ---- */

Ex:

// Prg for factorial of given number

/* This Program accepts a number from user and Calculate factorial of it and also display result*/



'C' Program Structure

```
// Program for Sum of 2 numbers
#include <stdio.h>  /* It is called linkage
                       Section, where header files are
                       included */

Void main()  // This is start of Exemution
{
    int a=5, b=3, c;  /* 3 variables
                         declared with 2 variable
                         Values 5, 3 and result
                         will be stored in c */
    c=a+b;
}
```

Execution starts

Line 1 : main( )

{

1 {

2

3

⋮

}

Line 10 main( ) ← Start

{

}

}

}

{

}

{

}

Line 50 main( ) ←

{

}

C Tokens - 2

( ) {

void

main

'C' Token

- The smallest, individual element of a program is called as Token.
- There are six types of 'C' Tokens :
  1. Identifiers main x y z Printf
  2. Constants 7 9
  3. Keywords void int
  4. Strings "result is %d"
  5. Operators = , *
  6. Special Symbols / Delimiters ( ) { ; }

Ex:

```
void main()
{
  int x=7, y=9, z;
  z= x*y;
  Printf("result is %d", z);
}
```

C Tokens - 2

BYJU'S EXAM PREP

Identifiers

- Variable
- Function
- arrays
- Pointers

etc ---

- Name Of any Programming Element is Identifier
- 5 rules to be followed for assigning name in 'C' Language
- Name should start with a letter or underscore symbol.
- Name should not contain space. Byjus_Exam_Prep
- Name should not be a keyword.
- Name should not contain any symbol except under score.
- Length of name should not exceed 32 characters.

7x ✗ 27 ✓ XYZ ✓ _7x ✓

x&y ✗ xy@ ✗ x_y ✓ xy_ ✓ _xy ✓

C Tokens - 2

BYJU'S EXAM PREP

int If; ✓

int if; ✗

int iF; ✓

Keywords

- A word whose meaning / purpose is pre-defined.
- All letters of a keyword must be in lower case.
- Keywords are also called as Reserved words.
- There are 32 keywords. in 'C'

do

static

if

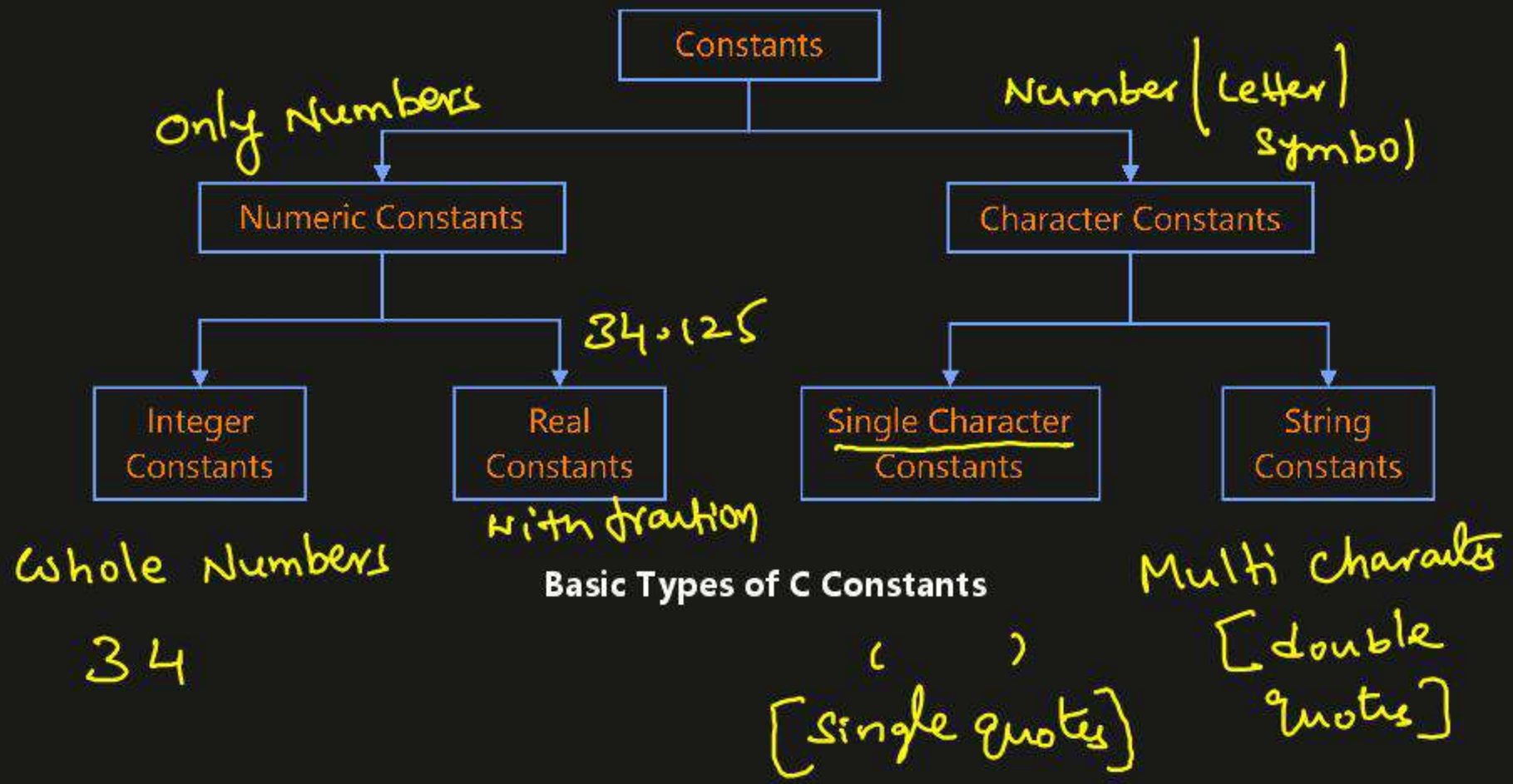
while

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile

'C' Keywords

BYJU'S EXAM PREP

Constants



Basic Types of C Constants

BYJU'S EXAM PREP

Integer Constants

265 : Decimal

0265 : Octal

0x265 : Hexa Decimal

Integer Constants

Decimal Integer

- Set of digits 0-9.
- Preceded by an optional - or + sign.
- Valid Examples :
  123
  -321
  +234
  0
  4569871
- Embedded spaces, commas, and non-digit characters are not allow between digits.
  Like :
  15 75
  20,000
  $10

Octal Integer

- Any combination of digits from 0-7 with a leading 0.
- Valid Examples:
  025
  0
  0123
  0456

Hexadecimal Integer

- Sequence of digits preceded by 0x or 0X.
- Digits 0-9 and Alphabets A-F
- Alphabets A-F represent the numbers 10-15.
- Valid Examples:
  0x1
  0X1
  0xca
  0X34

BYJU'S EXAM PREP

Operators

- An Operator Performs Operation On Operands.
- There are 3 types of operators : Unary, Binary and Ternary
- Unary Operators : Requires Single Operand
- Binary Operators : Requires Two Operands
- Ternary Operators : Requires Three Operands

Ex: a / b

Operation : Division

Operands : a, b

Operator : /

BYJU'S EXAM PREP

Unary Operators

Symbol	Meaning
++	Increment
--	Decrement
+	Sign plus
-	Sign minus
*	Pointer Indirection
&	Address Of
!	Logical NOT
~	Bitwise NOT
sizeof	sizeof
~~Symbol~~	~~Meaning~~

Unary Operators

+ : Sign → unary; Addition → Binary

− : Sign → unary; Subtraction → Binary

* : Pointer → unary; Multiplication → Binary.

Structure

• operator

→ operator

int i = -3;

More

• →

Postfix

Prefix, sizeof

operan

C Tokens - 2

BYJU'S EXAM PREP

++ and --

Prefix ++op / --op ++x, --y

Postfix op++ / op-- x++, i--

Pre increment(++variable)

It will increment variable value by 1 before assigning the value to the variable.

Syntax :

```
int var=11;
int out=++var; //out becomes 12
```

C Tokens - 2

++ and --

Post increment(variable++)

It will increment variable value by 1 after assigning the value to the variable.

Syntax :

```
int var=11;
int out=var++; //out becomes 11
```

12

11

C Tokens - 2

++ and --

Pre decrement(--variable)

It will decrement variable value by 1 before assigning the value to the variable.

Syntax :

```
int var=11;
int out=--var; //out becomes 10
```

BYJU'S EXAM PREP

++ and --

Post decrement(variable --)

It will decrement variable value by 1 after assigning the value to the variable.

Syntax :

```
int var=11;
int out=var--; //out becomes 11
```

C Tokens - 2

Binary Operators

- Arithmetic Operators
- Logical Operators
- Bitwise and Shift Operators
- Relational Operators
- Assignment Operators

BYJU'S EXAM PREP

Arithmetic Operators

2 = Q
4) 11
8
3 = R

2.75
4) 11
8
30
28
20
20
0

Ex:- a = 11 b = 4

c = a / b ; c = 2

d = a % b ; d = 3

Operator	Meaning of Operator
+	addition
-	subtraction
*	Multiplication
/	Division
%	Module operator

Quotient ← /

Remainder ← %

Modulus | mod

Arithmetic Operators

BYJU'S EXAM PREP

Logical Operators

Truth Values : True or False

Unary.

Non zero (±ve) : True

only zero : False

a = 11 b = -3

c = a && b

= (11) && (-3)

= T && T

= T == 1

$\boxed{c = 1}$

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Logical Operations Truth Table

BYJU'S EXAM PREP

Bitwise and Shift Operators

To be Contd ...

Truth table for OR, AND, and NOT operations

True = 1, False = 0

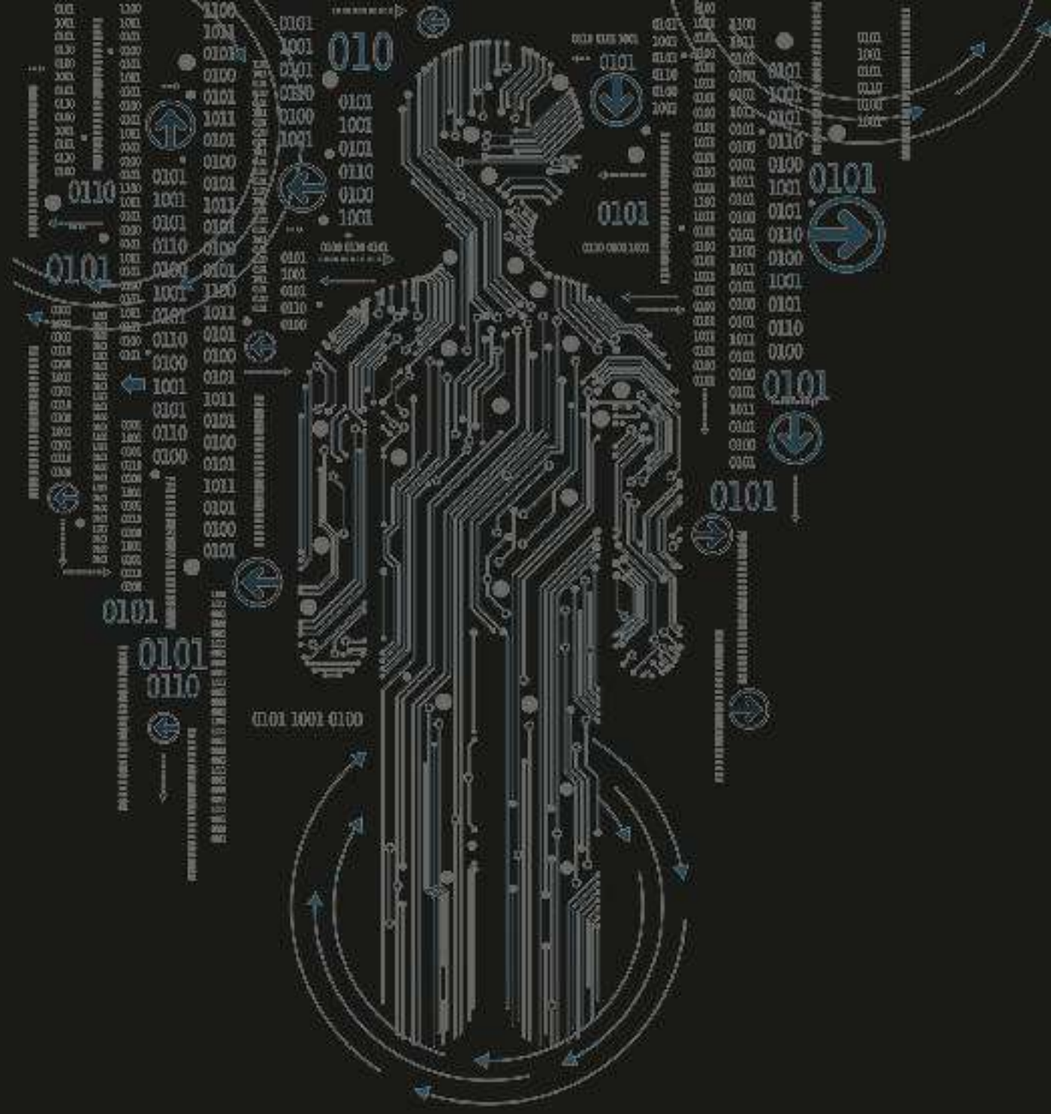
A	!A	B	!B	A || B	A&&B
0	1	0	1	0	0
1	0	0	1	1	0
0	1	1	0	1	0
1	0	1	0	1	1

Bitwise Operations Truth Table

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Data Types, I/O Statements



ABOUT ME

- Teaching Experience: 14+ Years
- Achievements: B.Tech, M.Tech, (Ph.D.)
- Achievements:
  - CS Senior Faculty
  - Mentored More Than 30,000+ Students
  - GATE Qualified
  - UGC-NET Qualified
  - SET Qualified
  - CISCO Certified

SATYA NARAYANA KAITEPALLI

Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

'C' Tokens :

- Identifiers, ✓
- Constants, ✓
- Strings, ✓
- Keywords, ✓
- Operators,
- Special Symbols.

When both Numerator & denominator are integers, then only results are integers.

$x = 17 \quad y = 5$

$a = x / y = 3$

$b = x \% y = 2$

$$\begin{array}{r} 3 = Q \\ 5 \overline{)17} \\ 15 \\ \hline (2) = R \end{array}$$

$x = \dfrac{17.0}{5}$ (or) $\dfrac{17.}{5.}$ (or) $\dfrac{17}{5.}$ $\Rightarrow$ result (Quotient) will be real number.

$y =$

$$\begin{array}{r} 3.4 = \text{Quotient} \\ 5 \overline{)17.} \\ 15 \\ \hline 20 \\ 20 \\ \hline (0) = \text{Remainder} \end{array}$$

$a = x / y \Rightarrow a = 3.400000$

$X = 17.46$ $\quad Y = 5.1$

$a = X/Y$

$$5.1 \overline{)17.46} \Rightarrow 51 \overline{)174.6}$$

3.---- = Quotient

153.0

21.6

0

if either X or Y or both are float type,

then $b = X \% Y \Rightarrow$ Error.

NOTE: "%" (mod) operator can be used only with integers.

Numerator x sign	Denominator y sign	x / y sign	x ·/· y sign
+	+	+	+
+	−	−	+
−	+	−	−
−	−	+	−

NOTE: ·/· operator will always consider Numerator sign for result (remainder)

$a = +23 \qquad b = -4$

$c = a / b \qquad c = -5$

$d = a \cdot/\cdot b \qquad d = +3$

$$\begin{array}{r|l} & -5 = Q \\ \hline -4 & +23 \\ & \underline{20} \\ & (+3) = R \end{array}$$

BYJU'S EXAM PREP

Bitwise and Shift Operators

1 == True 0 == False

Truth table for OR, AND, and NOT operations

True = 1, False = 0

A	!A (~A)	B	!B (~B)	A || B (A | B)	A&&B (A & B)
0	1	0	1	0	0
1	0	0	1	1	0
0	1	1	0	1	0
1	0	1	0	1	1

Bitwise Operations Truth Table

$a = 35 \quad b = 17$

$c = a \,\&\&\, b$

$c = \underline{35} \,\&\&\, 17$

$= T \,\&\&\, T$

$= T$

$\boxed{c = 1}$

$d = a \,\&\, b$

$d = 35 \,\&\, 17$

$35 : 0010\ 0011$

$17 : 0001\ 0001$

$0000\ 0001$

$d = 1$

$a = 35 \qquad b = 17$

$c = a \,||\, b$

$c = 35 \,||\, 17$

$= T \,||\, T$

$= T$

$\boxed{c = 1}$

Logical : T/F

$d = a \,|\, b$

$d = 35 \,|\, 17$

$35 = 0010 \;\; 0011$

$17 = 0001 \;\; 0001$

$(0011 \;\; 0011)_2$

$\boxed{d = 51}$

Bitwise = 1/0

$$A << n \cong A * 2^n$$

Bitwise and Shift Operators

$A = (14)_{10}$ $(0\ 0\ 0\ 0\ 1\ 1\ 1\ 0)_2$

$A << 1 \Rightarrow$

14 << 1	0	0	0	1	1	1	0	0	=	28	14*2

Empty boxes will be marked as 0

$A << 2 \Rightarrow$

14 << 2	0	0	1	1	1	0	0	0	=	56	28*2

above result will be shifted one more time

Empty boxes will be marked as 0

Left Shift Operation

After 4 left shifts = 1110 0000 == 224

|00000000 1110 0000|

A = 14

B = A << 3; 14 << 3 ≅ 14 * 2 * 2 * 2

LSB

$(14)_{10} = (0000\ 1110)_2$

MSB

			0	0	0	0	1	1	1	0
0	0	0	0	1	1	1	0	0	0	0

$(0111\ 0000)_2$ = 112 = 14 * 8

14×2^3

$$\boxed{A << n == A * 2^n}$$

BYJU'S EXAM PREP

Bitwise and Shift Operators

$14 >> 2 == 14 / 2^2 = 14 / 4 = 3$

$A >> n \cong A / 2^n$

A =

14	0	0	0	0	1	1	1	0

A >> 1

14 >> 1	0	0	0	0	0	1	1	1	=	7	14/2

Empty boxes will be marked as 0

A >> 2

14 >> 2	0	0	0	0	0	0	1	1	=	3	7/2

above result will be shifted one more time

Empty boxes will be marked as 0

Right Shift Operation

BYJU'S EXAM PREP

Relational Operators

Operator	Expression	Description
<	X < Y	X is less than Y
<=	X <= Y	X is less than or equal to Y
>	X > Y	X is greater than Y
>=	X >=Y	X is greater than or equal to Y
==	X == Y	X is equal to Y
!=	X != Y	X is not equal to Y

≤

≥

* Equal to

not Equal to

Relational Operators in 'C'

BYJU'S EXAM PREP

Assignment Operator

LHS = RHS

== vs =

a = 13 b = 11

c = a == b; c = False = 0

d = a = b; a = 11

d = 11

Assignment Operator	Shorthand operation
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

Assignment Operator

$$\boxed{\begin{array}{l} \text{LHS operator} = \text{RHS} \\ \cong \text{LHS} = \text{LHS operator RHS} \end{array}}$$

LHS op = RHS

Ex. ① $a + = 4 \cong a = a + 4$

Ex. ② $a = 4 \quad b = 5 \quad c = 2 \quad d = 7;$

$d - = a - b + c$ $\qquad$ d value = ?

$\cong d = d - [(a - b + c)] \cong d = d - a + b - c$

$= 7 - 4 + 5 - 2 = 6$

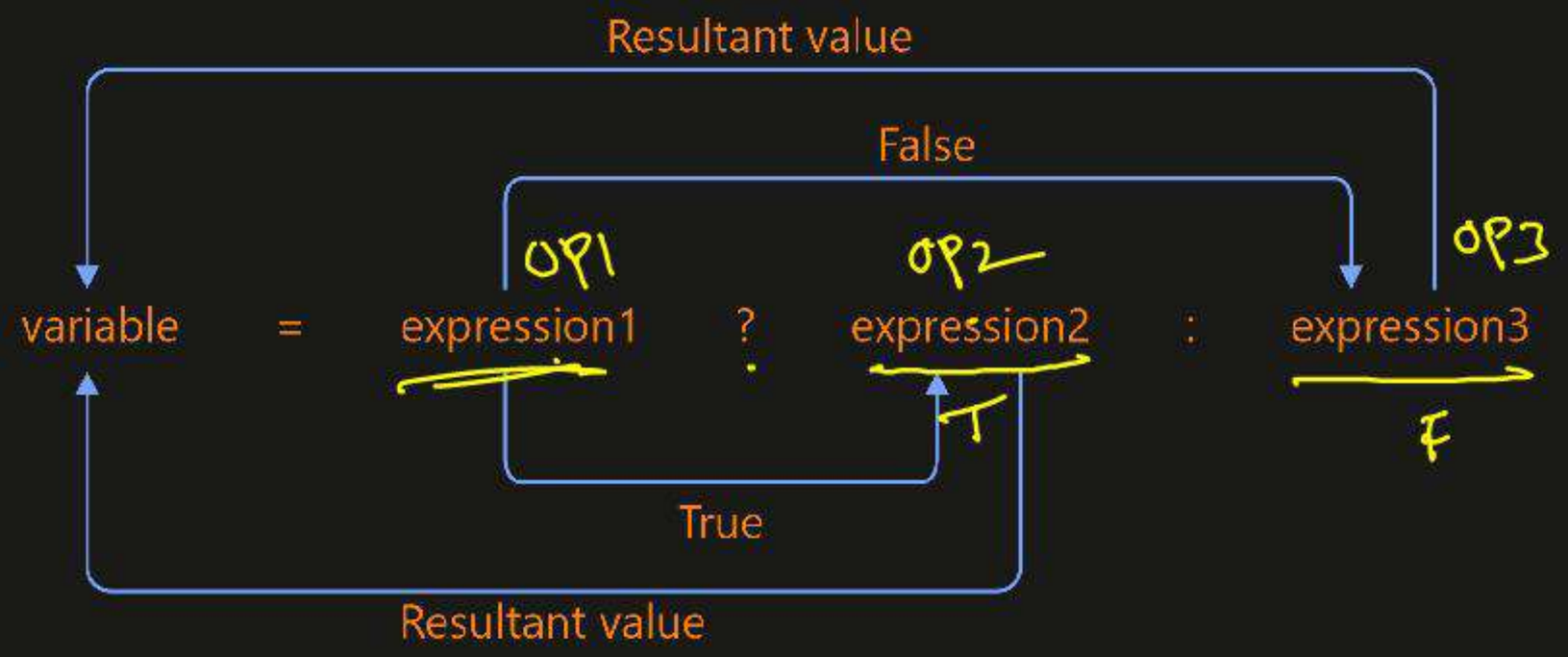
BYJU'S EXAM PREP

Ternary Operator

[3 operands]

Operand 1 ? operand 2 : Operand 3

? :



Ternary Operator

$a = 7 \quad b = 5$

$$c = \underbrace{(a > b)}_{op1} \; ? \; \underbrace{a}_{op2} \; : \; \underbrace{b}_{op3} ;$$

$$\underbrace{(7 > 5)}_{True} \longrightarrow \Rightarrow c = a \Rightarrow c = 7$$

$a = 7 \quad b = 5 \quad c = -2$

$$d = a \; ? \; b \; ? \; c \; ? \; a \; : \; b \; : \; c \; : \; a ;$$

a

$d = a$

$d = 7.$

$b \; ? \; a : c == a$

$a \; ? \; a : a$

BYJU'S EXAM PREP

Operator Precedence and Associativity

When Priority is same, then go with associativity.

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++--(type)* &sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right

BYJU'S EXAM PREP

Unary, Ternary, Assignment
R to L

Operator Precedence and Associativity

Category	Operator	Associativity
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Ternary

Logical

U A S R B L T A C — comma.

Unary
Arithmetic
Shift
Relational
Bitwise
Ternary
Assignment

A	B	A && B	A || B
1 | T	1 | T	1 == B	1 = A
		0 == B	1 = A
1 | T	0 | F		
0 | F	0 | F	0 == A	0 = B
0 | F	1 | T	0 == A	1 = B

→ short circuiting.

c = 1

Non zero/T

a++ ⇒ a=6

a = 5 b = 7

c = a++ || b--;

Neglected

c = op1 || op2

Neglected

a = 6 ✓

b = 7 ✓

c = 1 ✓

|| (OR)

Evaluate op2 only if op1 == 0

BYJU'S EXAM PREP

Data Types in 'C'

To be contd ...

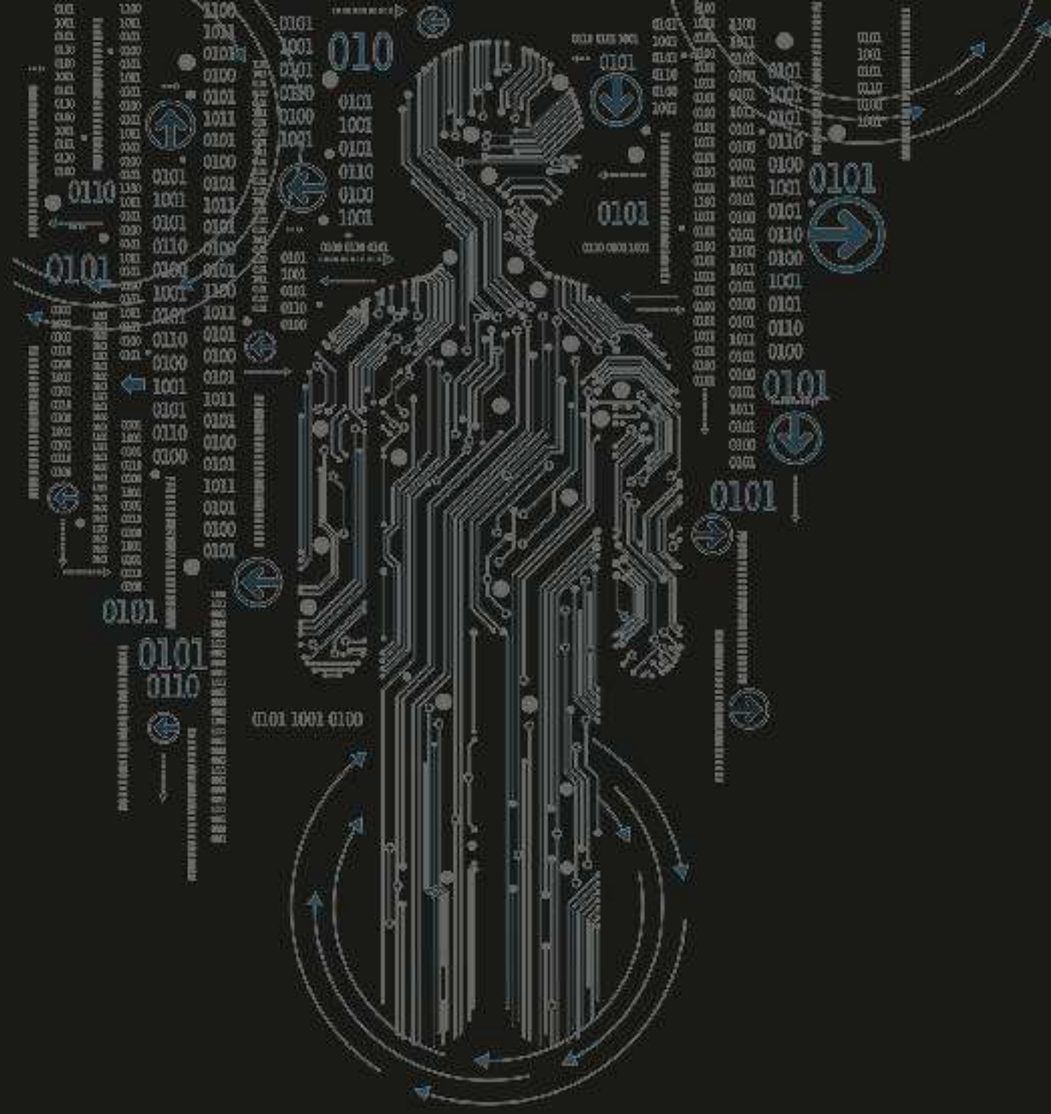
- Data Type defines:
  - a) Kind of Values accepted
  - b) Memory size required
  - c) Range of values supported by a variable.

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Control Statements



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Binary operators – Logical, Arithmetic, Bitwise, shift, Relational, Assignment.

Ternary – ? : (conditional)

Operator Precedence & Associativity

(U AS RBL TAC)

R to L

BYJU'S EXAM PREP

Data Types in 'C'

- Data Type defines:
  - a) "Kind of Values accepted"
  - b) Memory size required
  - c) Range of values supported by a variable.

Mail id : Xyz . 123 @ gmail . com

Variable | Constant

Variable: whose value keep changing

Constant: whose value does not change.

ATM
- Name
- A/c number
- Branch
- Balance amount
- Bank
- Pin number (0000 – 9999)
- Gender → character M/F

Max with draw limit 2,00,000/-

String

Whole number

String

float

Integer

– 123

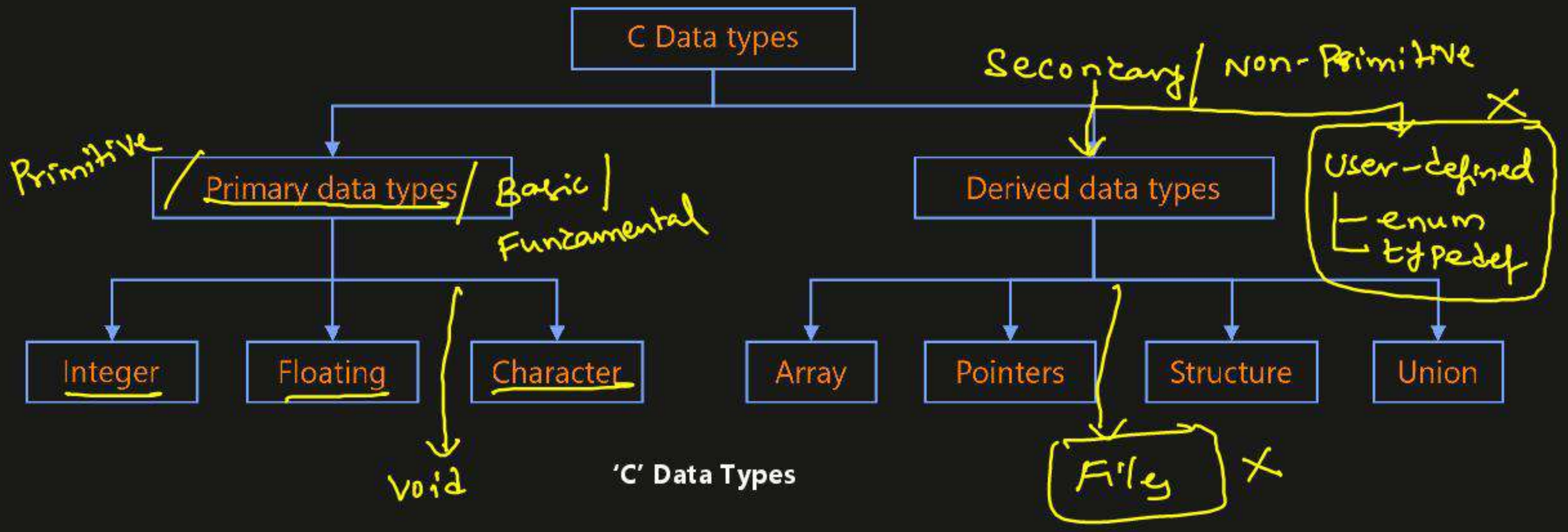
– ABC

ABC _XYZ

– ∞ to ∞

BYJU'S EXAM PREP

Data Types in 'C'



'C' Data Types

BYJU'S EXAM PREP

Integer Data Type

A data which is a number without fractional part.

- Integer data types are used to define the variables taking integer values with or without constant values given to them. The keyword or the data type used to define the integer type data is 'int'.

int variables;

Ex: int x, y, z;
int x=3, y=4, z;

- There are other data types like 'short' and 'long' used to define integer values but they have different ranges.
- The size of the integer data type is "compiler dependent".

16-bit Processor :	16 bits	2 Bytes
32-bit Processor :	32 bits	4 Bytes
64-bit " .	64-bits	8 Bytes

* 4 Bytes

Type Qualifiers : short, long, signed, unsigned

short int x; long int y; signed int z; unsigned int p;

BYJU'S EXAM PREP

Integer Data Type

Ex: days 28 to 31 — 8 bits | 1 Byte

i/p	o/p
30	April, June, sep, Nov
31	Jan, March, May, July, Aug, Dec
28 (or) 29	Feb

- The short integer can be used in places where small values and little storage space is required. It can boost up the runtime because it uses less space. It is declared by using the keyword 'short' or 'short int'. Ex: Accept Subject wise Marks to Print grade. var : Marks : 0 --- 100
- The long integer gives us a long range or a bigger size compared to 'short' but it can cause our program to take more time for execution because of the storage size it offers. It is declared by using the keyword 'long' or 'long int'. The range for a long integer is :

–2147483648 to +2147483647.

int	long int	short
2 Bytes	4 Bytes	1 Byte
4 Bytes	8 Bytes	2 Byte

$(100)_{10} = (0110\ 0100)_2$ — 8 bits | 1 Byte Enough.

BYJU'S EXAM PREP

Integer Data Type

Marks_in_GATE (±)

Marks_in_Exam (+ve only)

- We have another classification of the integer data type: signed and unsigned int.
- In case we need to take positive values only, then we can use unsigned int. Its range is 0 to 4294967295. This is because the leftmost bit is free and does not need to store the sign of the number. So, we get more storage space i.e. double on the positive side.
- By default, the signed type is declared and we do not need to use signed. Signed int works like an int. Its range is the same as int.

BYJU'S EXAM PREP

1 Byte == 8 bits

Character Data Type

: Letter (or) Number (or) Symbol.

a --- z | A --- Z 0 --- 9 @, +, -, $, <, &, ^

- Character data types are used to define variables taking one character as its value. The keyword used for character data type is 'char'. Here is how we declare character variables:

0=48 2=50 1=49

```
char ch, ch1='A', ch2=67;
```

a=97 A=65 b=98 B=66

- Here, in ch1 variable, we store 'A', i.e., the binary equivalent of the ASCII value of A(=binary of decimal 65) gets stored. And ch2 variable stores the value 67(ASCII for 'C'). So these are actually two ways of initializing a character value.

$=36 @=64 Space=32

- Like integers, here we have signed and unsigned character values. The signed char is equivalent to char. For a signed char the range is -128 to +127. Whereas, for an unsigned char the range is from 0 to 255.

Type Qualifier: signed (±ve), unsigned (+ve).

BYJU'S EXAM PREP

Float Data Type

: A number with fractional Part : Precision

→ single Precision : float → double Precision : long float == double

- Float and double data types are used to define variables that take up a decimal value or an exponential value. The keyword used for float and double data type are 'float' and 'double' respectively.

1.23 e 4 ⇒ $1.23 * 10^4$ ⇒ 12300

- Float has a range of –3.4e38 to +3.4e38 and its size is 4 bytes.
- Double has a range of -1.7e308 to +1.7e308 and its size is 8 bytes.

float : 4 Bytes
double : 8 Bytes
long double : 10 Bytes

- Another data type that is offered by programming languages is 'long double' which has a range of –1.7e4932 to +1.7e4932 and its size is 10 bytes. Here is how we declare float and double variables:
- Float type can have a 6-digit precision while double can have up to 14-digits.

BYJU'S EXAM PREP

Data Types Range

"i" bits Space → unsigned : 0 to $2^i - 1$
→ signed : $-2^{(i-1)}$ to $+2^{(i-1)} - 1$

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

$i = 8$ $-2^{(8-1)}$ to $2^{(8-1)} - 1$
$= -2^7$ to $(2^7 - 1)$
$= -128$ to $+127$

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

-1.e also

Range Of Values

$i = 2$ bits

0 =	0	0
1 =	0	1
2 =	1	0
3 =	1	1

Total 4 Possibilities 0 to 3

$i = 3$ bits

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

Total 8 Possibilities 0 to 7

i bits : Total 2^i Possibilities

Unsigned : 0 to $[2^i - 1]$

Signed : $\left[-\frac{2^i}{2} \text{ ----- } -1\right]\left[0 \text{ ---- } \left(\frac{2^i}{2} - 1\right)\right] = \left[-2^{i-1} \text{ --- } 2^{(i-1)} - 1\right]$

-ve

BYJU'S EXAM PREP

I/O Statements

Console : Interface
- Monitor o/p console
- Keyboard i/p console

These can be classified as follows:

- Console I/O functions: These functions allow us to receive input from the input devices like keyboard and provide output to the output devices like the Visual Display Unit.
- File I/O functions: These functions allow us to access the hard disk or floppy disk to perform input and output.

BYJU'S EXAM PREP

Console I/O functions

A console comprises the VDS and the keyboard. The Console Input and Output functions can be classified into two categories:

- Formatted console I/O functions: These functions allow the user to format the input from the keyboard and the output displayed in the desired manner. i/p scanf(), printf() o/p
- Unformatted console I/O functions: These functions do not allow the user , to process data by formatting. [as it is] getch(), gets(), i/p

  Putchar(), puts() o/p

BYJU'S EXAM PREP

sizeof : memory allocated in Bytes

strlen() : length of string
⇒ returns index of first NULL

Console I/O functions

- These various formatted and unformatted console I/O functions are shown in the figure below:

Formatted Functions		
Type	Input	Output
char	scanf()	printf()
int	scanf()	printf()
float	scanf()	printf()
string	scanf()	printf()

String : SATYA : No. of charaters = 5

Memory :

0	1	2	3	4	5
S	A	T	Y	A	\0

NULL charater [End of string]

sizeof(SATYA) : 6 Bytes
strlen(SATYA) : 5

Unformatted Functions		
Type	Input	Output
char	getch() getche() getchar()	putch() putchar()
int	-	-
float	-	-
string	gets()	puts()

I/O Functions

Strings : format specifier : %s

[& = = Address of]

Formatted Console i/p Function – scanf()

- scanf() is the formatted console input function which reads formatted input from stdin(standard input). It can read any integer, float, character, string, etc data from the user.
- The syntax of using scanf() is as follows:

```
scanf("format specifier" ,arguments address);
```

int x;

scanf("%d", &x)

d = decimal integer

- Let us consider an example: scanf("%d", &num);
- In this example, the %d is the format specifier for an integer hence, num stores an integer value. &num indicates the address of num where the value is to be stored under the variable name 'num'.

Ex: scanf("%s", &name); SATYA NARAYANA : i/p

- Note:- One disadvantage of scanf() when taking string inputs is that it will ignore the string that has been entered after a blank space. Thus, scanf() does not take multi-word string inputs. To take multi word string inputs we should use gets() method.

But SATYA is taken

BYJU'S EXAM PREP

Formatted Console o/p Function – printf()

- printf() is the formatted console output function which prints the formatted output to the stdout(standard output). It can display integers, floating point values, characters, string, etc as indicated by the user. The syntax of using printf() is as follows:

```
printf("text");
```

- This shall simply print "text" on the output screen. Any other textual data can be written within double quotes and displayed using printf() function. In order to print any variable value along with text we have the following format:

```
printf("text <format specifier>",variable);
```

```
int fact;
Printf("%ld",fact);
Printf("Factorial is %ld",
fact);
```

- This shall print the value of the variable in the format specified by the format specifier.
- For example, printf("marks: %d",marks); This has the format specifier as %d which stands for integer data and it shall print the marks in integer format.

BYJU'S EXAM PREP

Un Formatted Console I/O Functions

- The unformatted console input/output functions deal with a single character or a string of characters. Let us see how all these unformatted functions work:

Function	Description
getch()	This function returns the character that was typed last or was typed most recently. It does not display(echo) the character on the screen. It is present in the header file <conio.h>.
getche()	This function is the same as *getch()*. The only difference is that it echoes(displays) the most recently used character. It is also present in <conio.h>.
getchar()	*getchar()* is a macro that works in a similar manner as *getch()* and also displays(echoes) the character but it needs the user to press the Enter key after the character. It is present in the header file <stdio.h>.
fgetchar()	*fgetchar()* works in the same manner as *getchar()* . But *fgetchar()* is a function.It is also present in <stdio.h>.

BYJU'S EXAM PREP

Un Formatted Console I/O Functions

Function	Description
putch()	This function prints a single character on the console.
putchar()	This function works in the same way as *putch()*.
fputchar()	This function works in the same way as *putch()*.
gets()	It takes a string input(single-word or multi-word) from the user. It terminates when the Enter key is pressed.
puts()	It is used to print the string to the console.

BYJU'S EXAM PREP

ATM

Pin correct — Dispense cash

Pin wrong — Block | Error

Control Statement ?

- A single statement or group of statements, that can change the order of execution is called as control statement.
- There are 3 forms of control statements:
  a) Conditional statements — if, ifelse, switch
  b) Iterative statements : do-while, while, for
  c) Jumping statements — break, goto, continue

```
int a,b,c;
1 Printf(" Hai");
2 Pf("Hello");
3 Pf("Bye");
  c=a+b;
  Sf("  ");
```

o/p will be as in the order we write

Conditional Statements

To be contd...

- Conditional Control (Selection Control or Decision Control) :- In conditional control, the execution of statements depends upon the condition-test. If the condition evaluates to true, then a set of statements is executed otherwise another set of statements is followed.
- This control is also called Decision Control because it helps in making decision about which set of statements is to be executed.

BYJU'S EXAM PREP

Conditional Statements

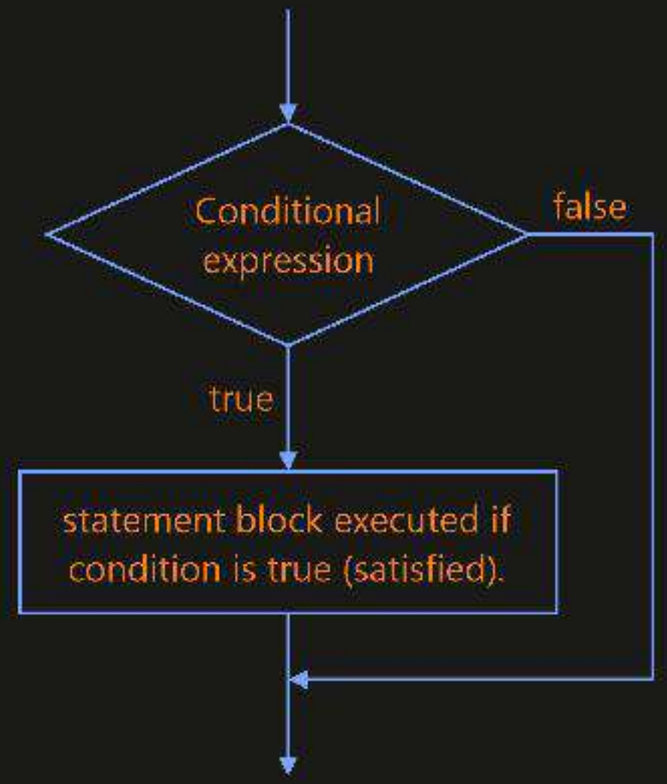
Decision control structure in C can be implemented by using:-

1. If statement
2. If-else statement
3. Nested if else statement
4. else-if ladder
5. case control structure
6. conditional operator

if statement

- This is the simplest form of decision control statement. In this form, a set of statements are executed only if the condition given with “if” evaluates to true.
- syntax :-

```
if(condition)
{
Statements ;
}
```



Flow chart : if

BYJU'S EXAM PREP

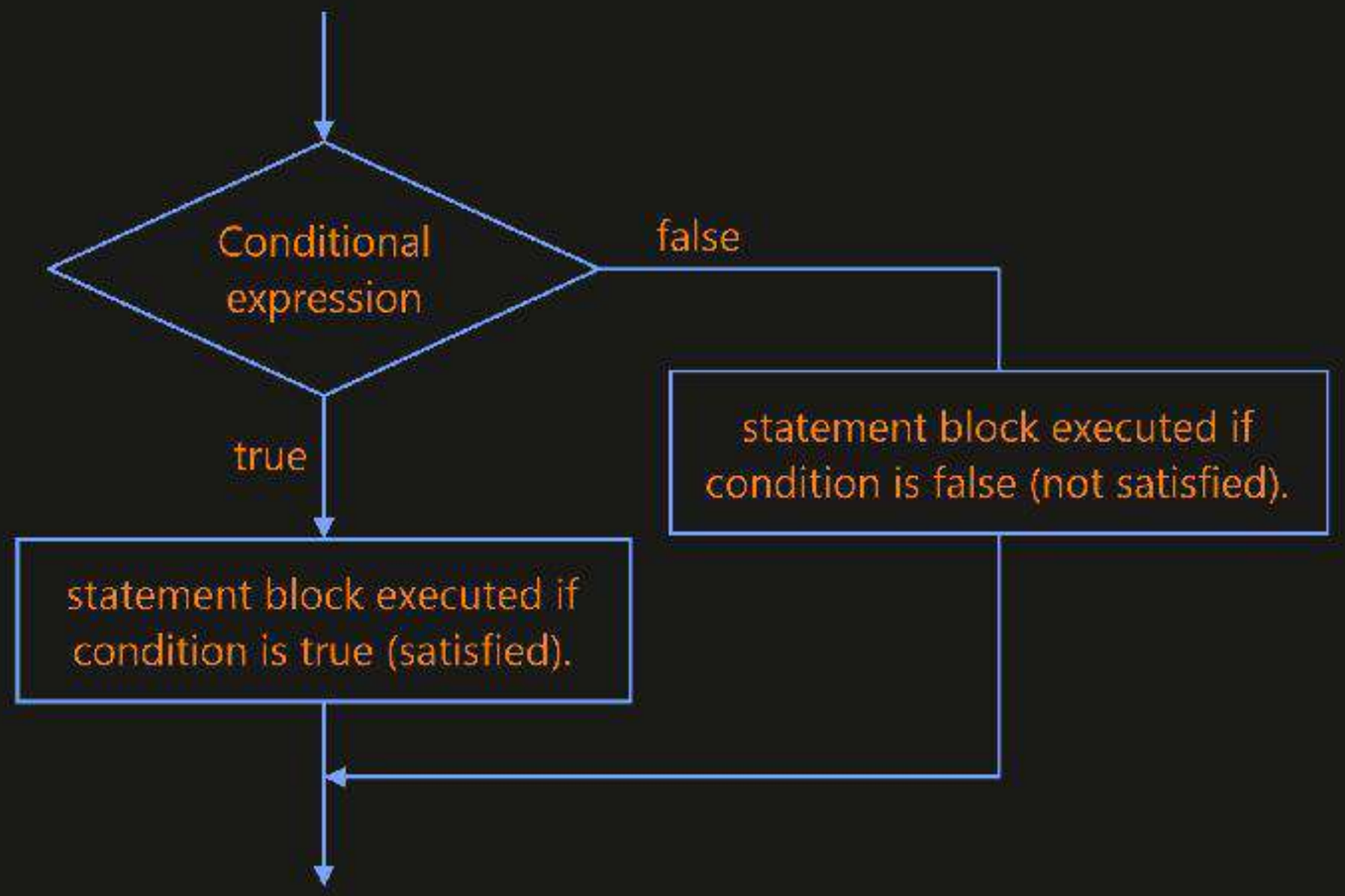
if-else

if- else statement:- This is a bi-directional control statement. This statement is used to test a condition and take one of the two possible actions. If the condition evaluates to true then one statement (or block of statements) is executed otherwise other statement (or block of statements) is executed.

The general form and flow chart of if-else statement is as under:-

```
if(condition)
{
block of statements;
}
else
{
block of statements;
}
```

if-else



Flow chart : if - else

BYJU'S EXAM PREP

Nested if-else

If we have if-else statement within either the body of an" if "statement or the body of "else" the statement or in the body of both if and else, then this is known as nesting of if-else statement.

The general form of nested if-else statement is as follows:-

```
if(condition1)
{
if(condition2)
statements;
else
statements;
}
else
{
if(condition3)
Statements;
else
Statements;
}
```

BYJU'S EXAM PREP

Else-if Ladder

This is a type of nesting in which there is an if-else statement in every else part except the last else part. This type of nesting is called else if ladder.

The general syntax and flow chart of else if ladder is as follows:-

```
If(condition1)
Statement A;
else if(condition2)
Statement B;
else if(condition3)
Statement C;
else
Statement D;
```

BYJU'S EXAM PREP

Switch-case

- Switch case statements are a substitute for long if statements and has more flexibility and a clearer format than else-if ladder.
- This statement is used to select one out of the several numbers of alternatives present in a block.
- This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

BYJU'S EXAM PREP

Switch-case

The general syntax of switch case statement is as follows:-

```
switch(expression)
{
case constant1: statements;
case constant2: statements;
case constant3: statements;
……..……..……..
……..……..……..
case constantN: statements;
default : statement;
}
```

Here switch, case and default are keywords.

BYJU'S EXAM PREP

Switch-case

- The expression following the switch keyword must give an integer value. This expression can be any integer or character variable, or a function call that returns an integer.
- It can also be arithmetic, relational, logical or bitwise expression yielding an integer.
- It can be integer or character constant also.
- Data types long int and short int are also allowed.
- The constant following the case keyword should be of integer or character type. We cannot use floating point or string constant.

BYJU'S EXAM PREP

Iterative Statements

Iterations or loops are used when we want to execute a statement or block of statements several times.

The repetition of loops is controlled with the help of a test condition. The statements in the loop keep on executing repetitively until the test condition becomes false.

There are the following three types of loops:-

1. while loop
2. do-while loop
3. for loop

BYJU'S EXAM PREP

while loop

It is the fundamental looping statement in C. It is suited for the problems where it is not known in advance that how many times a statement or block of statements will be executed.

The general syntax of while loop is as under:-

```
while(condition)
{
Statement(s);
}
```

BYJU'S EXAM PREP

while loop

Firstly the condition given with while is evaluated. If the condition evaluates to true then the statements given in the body of the loop are executed. After the execution of all the statements the condition is again checked and if it again evaluates to true then the statements given in the body of the loop are again executed. In this way the statements are executed again and again until the condition given evaluates to false.

For terminating the loop, a termination condition is given in the loop body which makes the condition false at some point of time and as a result of which the loop terminates. If we do not give the termination condition then the loop goes on repeating again and again infinite times. Such loops are referred to as infinite loops.

BYJU'S EXAM PREP

do-while loop

The do-while statement is also used for looping. It is similar to while loop and is used for the problems where it is not known in advance that how many times the statement or block of statements will be executed.

However, unlike while loop, in case of do-while, firstly the statements inside the loop body are executed and then the condition is evaluated. As a result of which this loop is executed at least once even if the condition is initially false.

After that the loop is repeated until the condition evaluates to false. Since in this loop the condition is tested after the execution of the loop, it is also known as post test loop.

The general syntax of do-while loop is as follows:-

```
do
{
Statement(s);
}while(condition)
```

BYJU'S EXAM PREP

for loop

The general syntax of for loop consists of three expressions separated by semicolons. It is given as follows:-

```
for(expression1;expression2;expression3)
{
Statement(s);
}
```

Here the expression1 is the initialization expression,expression2 is the test expression and expression3 is the update expression.

- Expression1 is executed only once when the loop starts and is used to initialize the loop variables.
- Expression2 is a condition and is tested before each iteration of the loop.
- Expression3 is an update expression and is executed each time after the body of the loop is executed.

BYJU'S EXAM PREP

Jumping Statements

- Jump statements are used to transfer the control from one part of the program to another part.
- The various jump statements in C are as under:-

1. break statement
2. continue statement
3. goto statement

BYJU'S EXAM PREP

break

Break statement is used inside the loops or switch statement. This statement causes an immediate exit from the loop or the switch case block in which it appears.

It can be written as

```
break;
```

BYJU'S EXAM PREP

continue

The continue statement is used inside the body of the loop statement.

when we want to go to the next iteration of the loop after skipping some of the statements of the loop.

The continue statement is written as under:

continue;

The difference between break and continue is that when a break statement is encountered the loop terminates and the control is transferred to the next statement following the loop, but when a continue statement is encountered the loop is not terminated and the control is transferred to the beginning of the loop.

BYJU'S EXAM PREP

goto

It is an unconditional statement and it transfers the control to the another part of the program.

The goto statement is used as under:-

```
goto label;
.................
.................
.................
label:
Statement;
.................
.................
```

Here label is any valid C identifier and is followed by a colon.

BYJU'S EXAM PREP

goto

- The goto statement transfers the control to the statement after the label.
- If the label is placed after the goto statement then the control is transferred forward and it is known as forward jump.
- If the label is placed before the goto statement then the control is transferred backward and the jump is called backward jump.
- In the forward jump all the statements between the goto statement and the label are skipped .
- In case of backward jump all the statement between the goto statement and the label are executed.
- There should always be a statement after any label.

BYJU'S EXAM PREP

Examples

BYJU'S EXAM PREP

Q1. Consider the following C code. Assume that unsigned long int type length is 64 bits.

```
unsigned long int fun(unsigned long int n){
    unsigned long int i, j = 0, sum = 0;
    for (i = n; i > 1; i = i/2) j++;
    for ( ; j > 1; j = j/2) sum++;
    return(sum);
}
```

The value returned when we call fun with the input 2^{40} is _____ .

[GATE 2018]

BYJU'S EXAM PREP

Q2. Consider the following C program:

```
# include<stdio.h>
int main ( )
{
   int i, j, k = 0;
   j = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
   k –= – –j;
   for (i = 0; i < 5: i++)
   {
      switch (i + k)        {
         case 1:
         case 2: printf (“\ n%d”, i+k);
         case 3: printf (“\n%d”, i+k);
         default: printf (“\n%d”, i+k);        }
   }
   return 0;
}
```

The number of times printf statement is executed is _____.

[GATE 2015]

BYJU'S EXAM PREP

Q3. What will be the output of the following C program segment?

```
char inChar = 'A' ;
switch ( inChar ) {
case 'A' : printf ("Choice A\ n") ;
case 'B' :
case 'C' : printf ("Choice B") ;
case 'D' :
case 'E' :
default : printf ( " No Choice" ) ; }
```

[GATE 2012]

A. No Choice

B. Choice A

C. Choice A,
Choice B No Choice

D. Program gives no output as it is erroneous

BYJU'S EXAM PREP

Q4. Consider the following C program

```
main()
{
  int x, y, m, n;
  scanf ("%d %d", &x, &y);
  /* x > 0 and y > 0 */
  m = x; n = y;
  while (m != n)
  {
    if(m>n)
      m = m - n;
    else
      n = n - m;
  }
  printf("%d", n);
}
```

The program computes

[GATE 2004]

A. x + y using repeated subtraction.

B. x mod y using repeated subtraction.

C. The greatest common divisor of x & y.

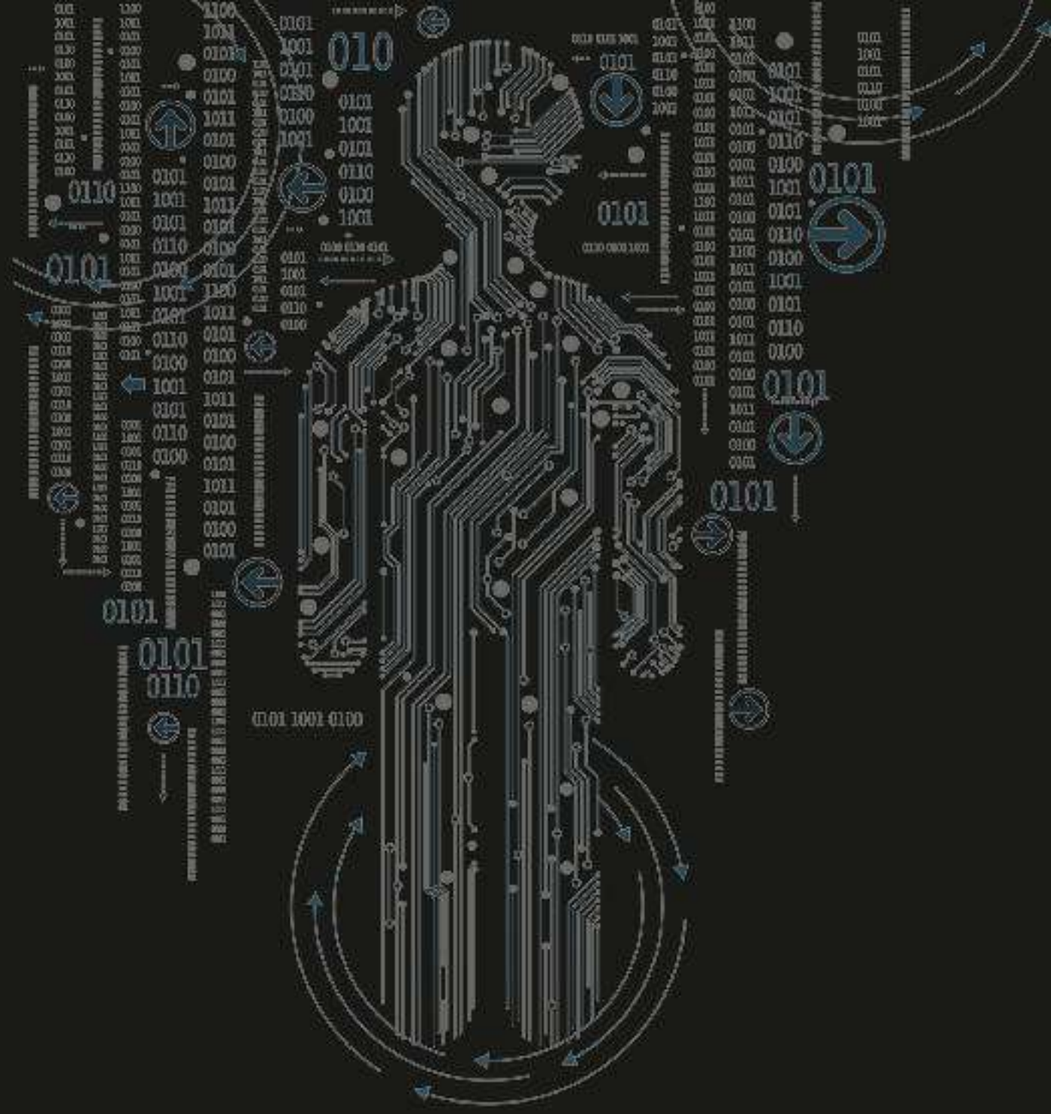
D. The least common multiple of x & y.

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Pointers



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

NOTE: Extra class Tomorrow @ 12PM

BYJU'S EXAM PREP

Last Class – Quick Revision

- Data Types
  - Primary : void, char, int, float
  - Secondary
    - Derived
    - User-defined

Range : Memory = i bits
- Unsigned 0 to $2^i - 1$
- Signed $-2^{(i-1)}$ to $+2^{(i-1)} - 1$

- I/O statements : Console I/O
  - Formatted : scanf(), printf()
  - Unformatted : getch(), putch() ...
- Control Statements : change order of Execution.

BYJU'S EXAM PREP

Conditional Statements

marks >= 40

Pass

a > 3

Pin == Correct

Dispense Cash

- Conditional Control (Selection Control or Decision Control) :- In conditional control, the execution of statements depends upon the condition-test. If the condition evaluates to true, then a set of statements is executed otherwise another set of statements is followed.
- This control is also called Decision Control because it helps in making decision about which set of statements is to be executed.

BYJU'S EXAM PREP

Conditional Statements

Decision control structure in C can be implemented by using:-

1. If statement
2. If-else statement , Nested if
3. Nested if else statement
4. else-if ladder
5. case control structure [switch]
6. conditional operator ? :

BYJU'S EXAM PREP

if statement

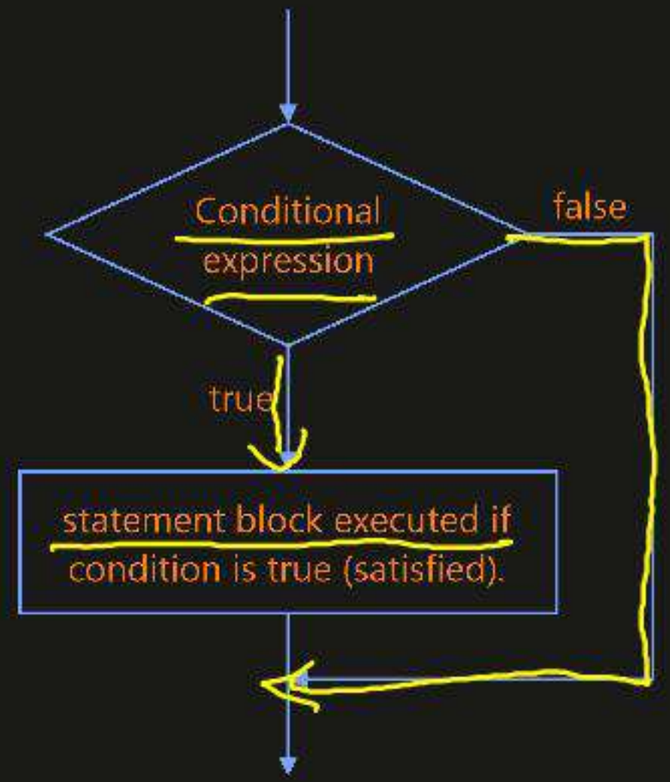
{ } => Block

- This is the simplest form of decision control statement. In this form, a set of statements are executed only if the condition given with "if" evaluates to true.
- syntax :-

```
if(condition)
{
Statements ;
}
```

if block

if single statement, then { } are optional



Flow chart : if

```
1. void main( ){
2.  int x, y;
3.  if( x>y) {   TRUE
4.      Printf("Hello");
5.      Printf("How are you");}
6.  Printf("Bye");  }
```

False

$x=5, y=3$

1, 2, 3, 4, 5, 6 lines Executed

$x=3, y=5$

1, 2, 3, 6 lines Executed.

```
1 Void main( ){
2 int x;
3 if( x)
4   Printf("Hello");
5 Printf("Bye"); }
```

$x=0$

1, 2, 3, 5

if(0)

False

$x=$ Non Zero

1, 2, 3, 4, 5

if(Non zero)

True

```
// To find Pass (or) Fail
  void main( )
  {
    int marks;
    scanf("%d", &marks);
     if (marks >= 40){
      Printf(" PASS");
      Printf(" CONGRATULATIONS"); }
      Printf(" FAIL");
      Printf(" BETTER LUCK NEXT TIME");
  }
```

i/p

marks = 60

60 >= 40

TRUE

o/p :

PASS

CONGRT--

FAIL

BETTER---

BYJU'S EXAM PREP

if-else

if- else statement:- This is a bi-directional control statement. This statement is used to test a condition and take one of the two possible actions. If the condition evaluates to true then one statement (or block of statements) is executed otherwise other statement (or block of statements) is executed.

if

else

The general form and flow chart of if-else statement is as under:-

```
if(condition)
{
block of statements;
}
else
{
block of statements;
}
```

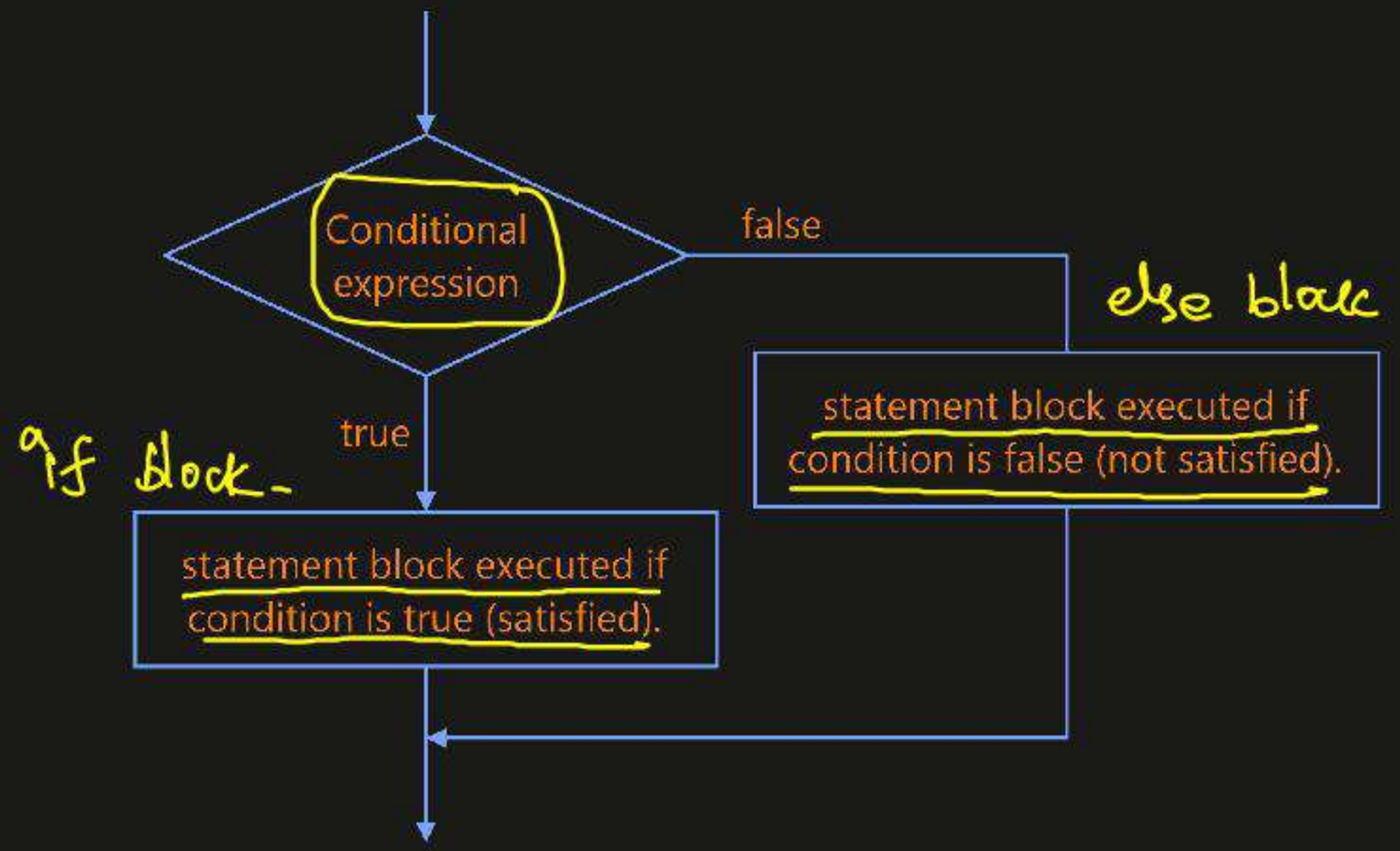
TRUE

FALSE

NOTE: "else" should be immediately after if.

BYJU'S EXAM PREP

if-else



Flow chart : if - else

```
if (x)
{ Printf("Hai");
}
Printf("Hello");
else
{ Printf("Bye");
}
```

o/p: ERROR

"Misplaced else"

```
if (x >= 40)
  Pf("PASS");
else
  Pf("FAIL");
```

i/p	o/p
55	PASS
40	PASS
20	FAIL
90	PASS
39	FAIL

```
int a, b, c;
Scanf("%d %d %d",
      &a, &b, &c);
if (a, b, c)
  Printf("Hai");
else
  Printf("Bye");
```

Decision making

i/p a	b	c	o/p
0	1	0	Bye
0	0	1	Hai
1	1	0	Bye
1	1	1	Hai

```
int a = 6, b = 3, c = -2;
if (a++, --b, c+2)
Printf("Hai");
else
Printf("Bye");
Printf("%d %d %d", a, b, c);
```

It is Not $c = c+2$ (or) $c+=2$

$\Rightarrow$ if (5, 3, 0), Then a++ happens

False

o/p: Bye 6 3 -2

$a = 1 + 3$

$y = a++ + a++;$

$1 + 2$

$y = 1 + 2 = 3$

BYJU'S EXAM PREP

Nested if-else

If we have if-else statement within either the body of an" if "statement or the body of "else" the statement or in the body of both if and else, then this is known as nesting of if-else statement.

The general form of nested if-else statement is as follows:-

```
if(condition1)
{
if(condition2)
statements;
else
statements;
}
else
{
if(condition3)
Statements;
else
Statements;
}
```

/* Find Maximum of 3 given numbers */

```
void main( )
{
    int a, b, c ;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>b)
    {
        if (a>c)
            Pf ("a is big");
        else
            Pf ("c is big");
    }
    else
    {
        if (b>c)
            Pf ("b is big");
        else
            Pf ("c is big");
    }
}
```

3>5 False

5>2 True

a	b	c
3	5	2
	b is big	
2	5	10
	c is big	
5	5	5
(5>5) False		
(5>5) False		
	c is big.	

BYJU'S EXAM PREP

Else-if Ladder

This is a type of nesting in which there is an if-else statement in every else part except the last else part. This type of nesting is called else if ladder.

The general syntax and flow chart of else if ladder is as follows:-

```
If(condition1)
Statement A;
else if(condition2)
Statement B;
else if(condition3)
Statement C;
else
Statement D;
```

```
if (rollno ==1)
    Pf("student 1");
else if (rollno==2)
    Pf("student 2");
else if (rollno==3)
    Pf("student 3");
    :
```

BYJU'S EXAM PREP

Switch-case

Multi-way selection statement.

- Switch case statements are a substitute for long if statements and has more flexibility and a clearer format than else-if ladder.
- This statement is used to select one out of the several numbers of alternatives present in a block.
- This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

BYJU'S EXAM PREP

Switch-case

The general syntax of switch case statement is as follows:-

```
switch(expression)
{
case constant1: statements;
case constant2: statements;
case constant3: statements;
.....................
.....................
case constantN: statements;
default : statement;
}
```

break;
break;
break;

when NO case is Matched

Ex:-

```
Switch(op)
{
Case 1: Pf("Hi"); break;
Case 2: Pf("Hello"); break;
Case 3: Pf("Bye"); break;
Case 4: Pf("Take care"); break;
default: Pf("SORRY");
}
```

Here switch, case and default are keywords.

BYJU'S EXAM PREP

Switch-case

* ① only int | char cases
* ② default, break are optional
* ③ cases can be random
* ④ cases can be Non-Contiguous
* ⑤ default can be any where.

- The expression following the switch keyword must give an integer value. This expression can be any integer or character variable, or a function call that returns an integer.
- It can also be arithmetic, relational, logical or bitwise expression yielding an integer.
- It can be integer or character constant also.
- Data types long int and short int are also allowed.
- The constant following the case keyword should be of integer or character type. We cannot use floating point or string constant.

```
int x;
scanf("%d", &x);
switch(x)
{
  Case 7: Pf("Blue");
  default: Pf("Black");
  Case 20: Pf("orange");
  Case 14: Pf("Red");
  Case 96: Pf("yellow"); break;
  Case 112: Pf("Purple");
  Case 5: Pf("Green");
}
```

30==7 False

Matched

30==20 False

30==14 False

30==96 False

30==112 False

30==5 False

x	o/p
7	Blue Black orange Red yellow
20	orange Red Yellow
5	Green
30	if there is No default No o/p.
	Black orange Red Yellow.
0	Black orange Red Yellow.

BYJU'S EXAM PREP

Iterative Statements

Iterations or loops are used when we want to execute a statement or block of statements several times.

The repetition of loops is controlled with the help of a test condition. The statements in the loop keep on executing repetitively until the test condition becomes false.

There are the following three types of loops:-

1. while loop
2. do-while loop
3. for loop

BYJU'S EXAM PREP

while loop

It is the fundamental looping statement in C. It is suited for the problems where it is not known in advance that how many times a statement or block of statements will be executed.

The general syntax of while loop is as under:-

```
while(condition)
{
Statement(s);
}
```

BYJU'S EXAM PREP

while loop

Firstly the condition given with while is evaluated. If the condition evaluates to true then the statements given in the body of the loop are executed. After the execution of all the statements the condition is again checked and if it again evaluates to true then the statements given in the body of the loop are again executed. In this way the statements are executed again and again until the condition given evaluates to false.

For terminating the loop, a termination condition is given in the loop body which makes the condition false at some point of time and as a result of which the loop terminates. If we do not give the termination condition then the loop goes on repeating again and again infinite times. Such loops are referred to as infinite loops.

BYJU'S EXAM PREP

do-while loop

The do-while statement is also used for looping. It is similar to while loop and is used for the problems where it is not known in advance that how many times the statement or block of statements will be executed.

However, unlike while loop, in case of do-while, firstly the statements inside the loop body are executed and then the condition is evaluated. As a result of which this loop is executed at least once even if the condition is initially false.

After that the loop is repeated until the condition evaluates to false. Since in this loop the condition is tested after the execution of the loop, it is also known as post test loop.

The general syntax of do-while loop is as follows:-

```
do
{
Statement(s);
}while(condition)
```

BYJU'S EXAM PREP

for loop

The general syntax of for loop consists of three expressions separated by semicolons. It is given as follows:-

```
for(expression1;expression2;expression3)
{
Statement(s);
}
```

Here the expression1 is the initialization expression,expression2 is the test expression and expression3 is the update expression.

- Expression1 is executed only once when the loop starts and is used to initialize the loop variables.
- Expression2 is a condition and is tested before each iteration of the loop.
- Expression3 is an update expression and is executed each time after the body of the loop is executed.

BYJU'S EXAM PREP

Jumping Statements

- Jump statements are used to transfer the control from one part of the program to another part.
- The various jump statements in C are as under:-

1. break statement
2. continue statement
3. goto statement

BYJU'S EXAM PREP

break

Break statement is used inside the loops or switch statement. This statement causes an immediate exit from the loop or the switch case block in which it appears.

It can be written as

```
break;
```

BYJU'S EXAM PREP

continue

The continue statement is used inside the body of the loop statement.

when we want to go to the next iteration of the loop after skipping some of the statements of the loop.

The continue statement is written as under:

continue;

The difference between break and continue is that when a break statement is encountered the loop terminates and the control is transferred to the next statement following the loop, but when a continue statement is encountered the loop is not terminated and the control is transferred to the beginning of the loop.

BYJU'S EXAM PREP

goto

It is an unconditional statement and it transfers the control to the another part of the program.

The goto statement is used as under:-

```
goto label;
...................
...................
...................
label:
Statement;
...................
...................
```

Here label is any valid C identifier and is followed by a colon.

BYJU'S EXAM PREP

goto

- The goto statement transfers the control to the statement after the label.
- If the label is placed after the goto statement then the control is transferred forward and it is known as forward jump.
- If the label is placed before the goto statement then the control is transferred backward and the jump is called backward jump.
- In the forward jump all the statements between the goto statement and the label are skipped .
- In case of backward jump all the statement between the goto statement and the label are executed.
- There should always be a statement after any label.

BYJU'S EXAM PREP

Q1. Consider the following C code. Assume that unsigned long int type length is 64 bits.

```
unsigned long int fun(unsigned long int n){
    unsigned long int i, j = 0, sum = 0;
    for (i = n; i > 1; i = i/2) j++;
    for ( ; j > 1; j = j/2) sum++;
    return(sum);
}
```

The value returned when we call fun with the input 2^{40} is _____ .

[GATE 2018]

BYJU'S EXAM PREP

Q2. Consider the following C program:

```
# include<stdio.h>
int main ( )
{
   int i, j, k = 0;
   j = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
   k –= – –j;
   for (i = 0; i < 5; i++)
   {
      switch (i + k)        {
         case 1:
         case 2: printf ("\ n%d", i+k);
         case 3: printf ("\n%d", i+k);
         default: printf ("\n%d", i+k);        }
   }
   return 0;
}
```

i = 0, 1, 2, 3, 4

The number of times printf statement is executed is _____.

[GATE 2015]

To be Contd...

BYJU'S EXAM PREP

Q3. What will be the output of the following C program segment?

```
char inChar = 'A' ;
switch ( inChar ) {
case 'A' : printf ("Choice A\ n") ;
case 'B' :
case 'C' : printf ("Choice B") ;
case 'D' :
case 'E' :
default : printf ( " No Choice" ) ; }
```

[GATE 2012]

A. No Choice

B. Choice A

C. Choice A,
Choice B No Choice

D. Program gives no output as it is erroneous

BYJU'S EXAM PREP

Q4. Consider the following C program

```
main()
{
  int x, y, m, n;
  scanf ("%d %d", &x, &y);
  /* x > 0 and y > 0 */
  m = x; n = y;
  while (m != n)
  {
    if(m>n)
      m = m - n;
    else
      n = n - m;
  }
  printf("%d", n);
}
```

The program computes

[GATE 2004]

A. x + y using repeated subtraction.

B. x mod y using repeated subtraction.

C. The greatest common divisor of x & y.

D. The least common multiple of x & y.

BYJU'S EXAM PREP

Pointer

- Pointer is a variable that holds address of another data item.
- A pointer can hold address of a variable or an array or a function or a structure or a file or another pointer also.
- Example:

BYJU'S EXAM PREP

Pointer Declaration and Initialization

- The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.
- Example:

```
int *a;//pointer to int

char *c;//pointer to char
```

BYJU'S EXAM PREP

Pointer Data Type

- As pointers hold addresses, address cannot be negative and fractional, So the data type of pointer will always be unsigned integer.
- While declaring a pointer, data type describes, what kind of data address, a pointer should hold.

BYJU'S EXAM PREP

Pointer Size

- As pointer holds address and as address is always unsigned integer type, size of pointer will always be integer size, while the integer size is compiler dependent.
- Example:

BYJU'S EXAM PREP

Pointer Arithmetic

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- Increment
- Decrement
- Addition
- Subtraction
- Comparison

BYJU'S EXAM PREP

Types Of Pointers

There are various types of pointers, that include:

- Void pointer
- Null Pointer
- Dangling Pointer
- Near Pointer
- Far Pointer
- Array Pointer
- Function Pointer
- Structure Pointer
- Pointer to Pointer etc.

BYJU'S EXAM PREP

Null Pointer

A Null Pointer is a pointer that does not point to any memory location. It stores the base address of the segment. The null pointer basically stores the Null value while void is the type of the pointer.

A null pointer is a special reserved value which is defined in a stddef header file. Here, Null means that the pointer is referring to the 0th memory location.

If we do not have any address which is to be assigned to the pointer, then it is known as a null pointer. When a NULL value is assigned to the pointer, then it is considered as a Null pointer.

Following are the applications of a Null pointer:

- It is used to initialize 0 pointer variable when the pointer does not point to a valid memory address.
- It is used to perform error handling with pointers before dereferencing the pointers.
- It is passed as a function argument and to return from a function when we do not want to pass the actual memory address.

BYJU'S EXAM PREP

Void Pointer

if we declare the int pointer, then this int pointer cannot point to the float variable or some other type of variable, i.e., it can point to only int type variable.

To overcome this problem, we use a pointer to void. A pointer to void means a generic pointer that can point to any data type.

We can assign the address of any data type to the void pointer, and a void pointer can be assigned to any type of the pointer without performing any explicit typecasting.

Syntax of void pointer

void *pointer name;

Declaration of the void pointer is given below:

void *ptr;

BYJU'S EXAM PREP

Pointer to Pointer

As we know that, a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable.

However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer).

The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.

The syntax of declaring a double pointer is given below.

```
int **p; // pointer to a pointer which is pointing to an integer.
```

BYJU'S EXAM PREP

Dangling Pointer

A pointer with an invalid address is known as a dangling pointer in C.

Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer.

In this case, the pointer is pointing to the memory, which is de-allocated. The dangling pointer can point to the memory, which contains either the program code or the code of the operating system.

BYJU'S EXAM PREP

Const Pointer

A constant pointer in C cannot change the address of the variable to which it is pointing, i.e., the address will remain constant. Therefore, we can say that if a constant pointer is pointing to some variable, then it cannot point to any other variable.

Syntax of Constant Pointer

<type of pointer> *const <name of pointer>;

The Declaration of a constant pointer is given below:

int *const ptr;

BYJU'S EXAM PREP

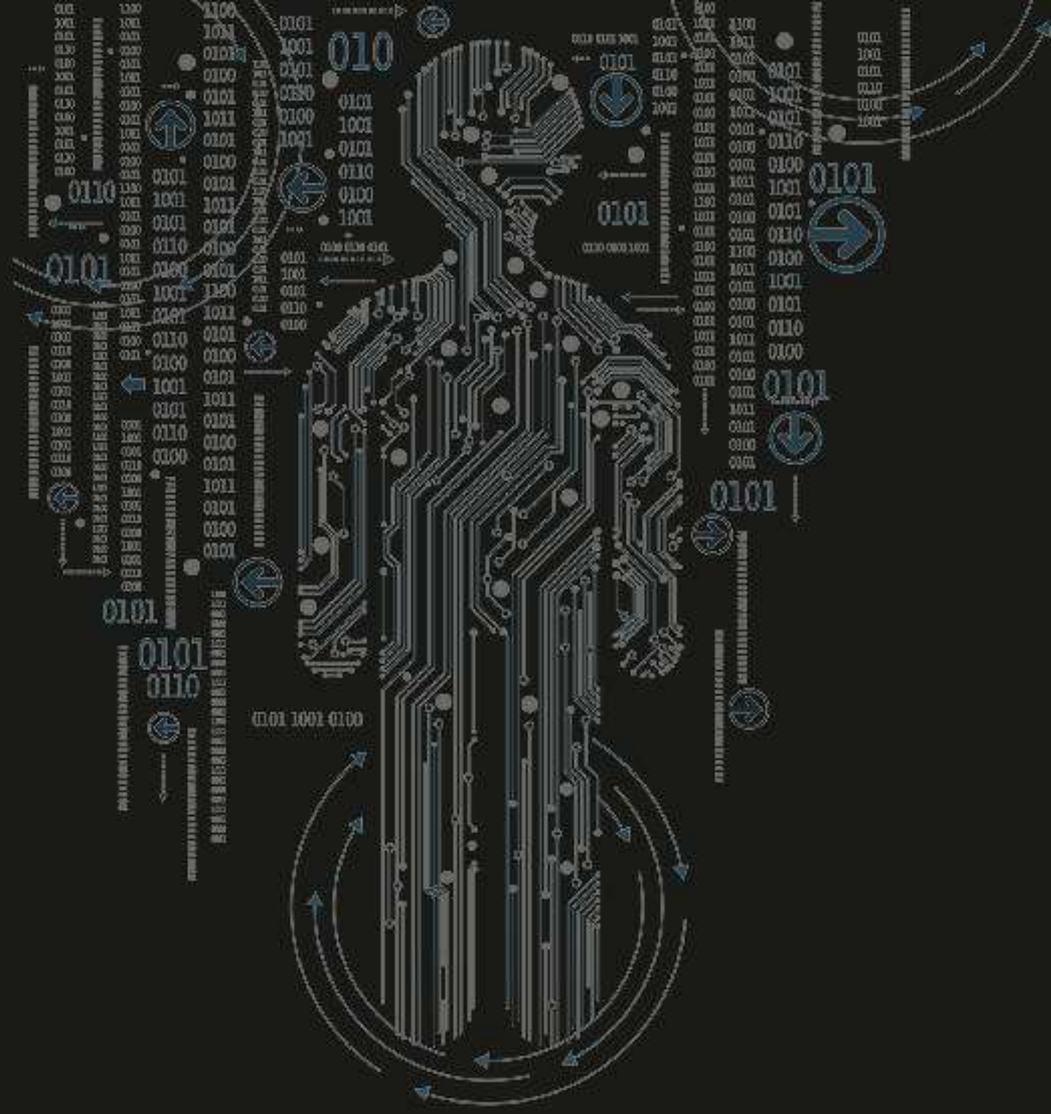
Examples

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Pointers & Arrays – 1



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

- Function : which Perform specific Sub Task
  - System-defined
  - User-defined
- Declaration, Definition, calling
- Prototype Declaration is Needed when?
- Caller, Callee function
- Function Prototypes : with/without arg, with/without r-value
- Formal arguments, Actual arguments
- Recursion : Caller == Callee
  - Direct
  - Indirect

A → A

A → B → C → A

BYJU'S EXAM PREP

Types Of Direct Recursion

Direct Recursion Can be implemented in either of following ways:

- Head Recursion
- Tail Recursion
- Nested Recursion
- Tree Recursion

BYJU'S EXAM PREP

Head Recursion

When the recursive call is the first or early statement in recursive function, then it is called head recursion.

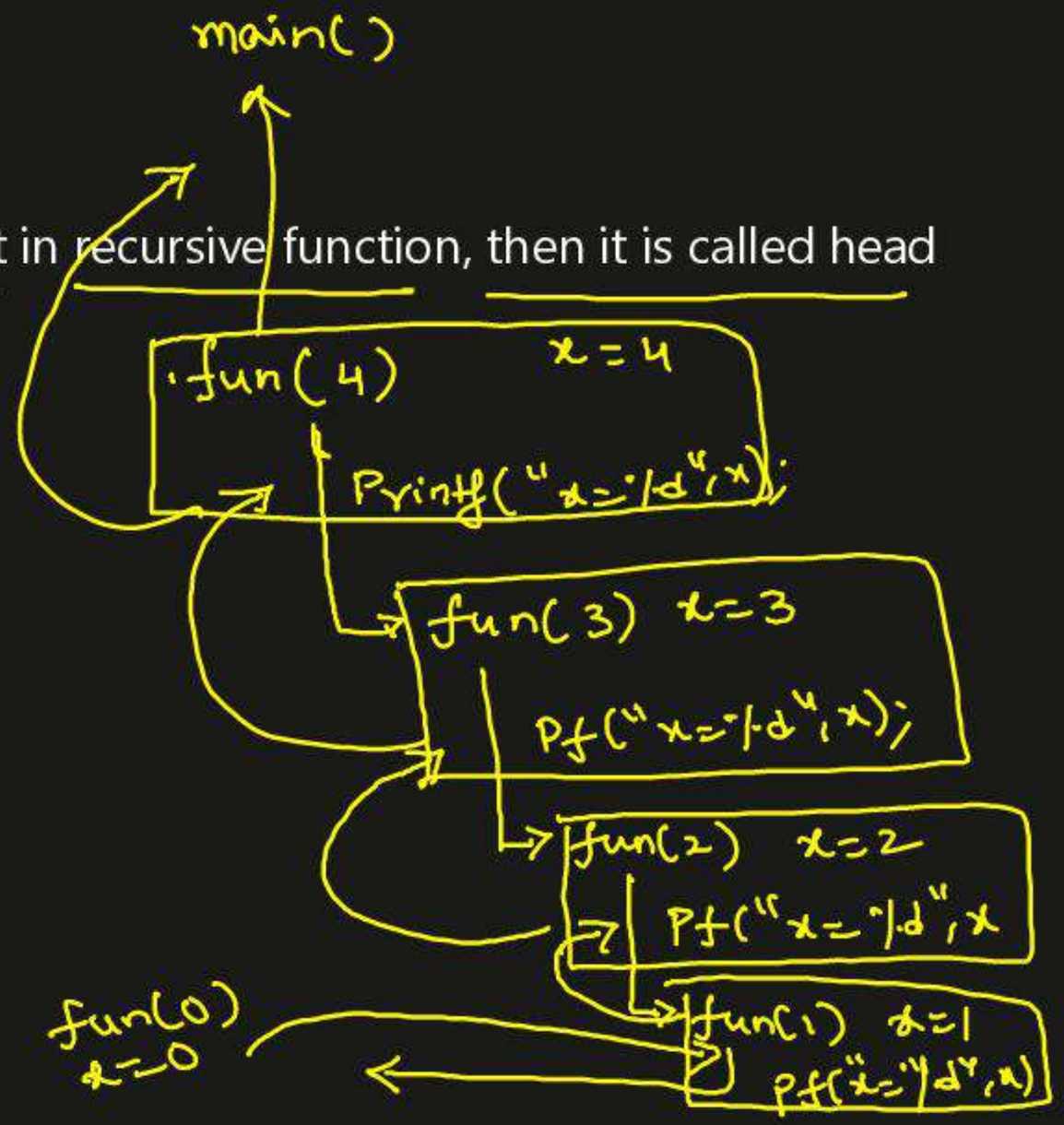
Example:

```
void/int fun(int x)
{
if(x==0) return;
else{
fun(x-1);
printf("x=%d",x);
}
}
```

Not executed yet

```
void main()
{
fun(4);
}
```

o/p: 1 2 3 4



BYJU'S EXAM PREP

Tail Recursion

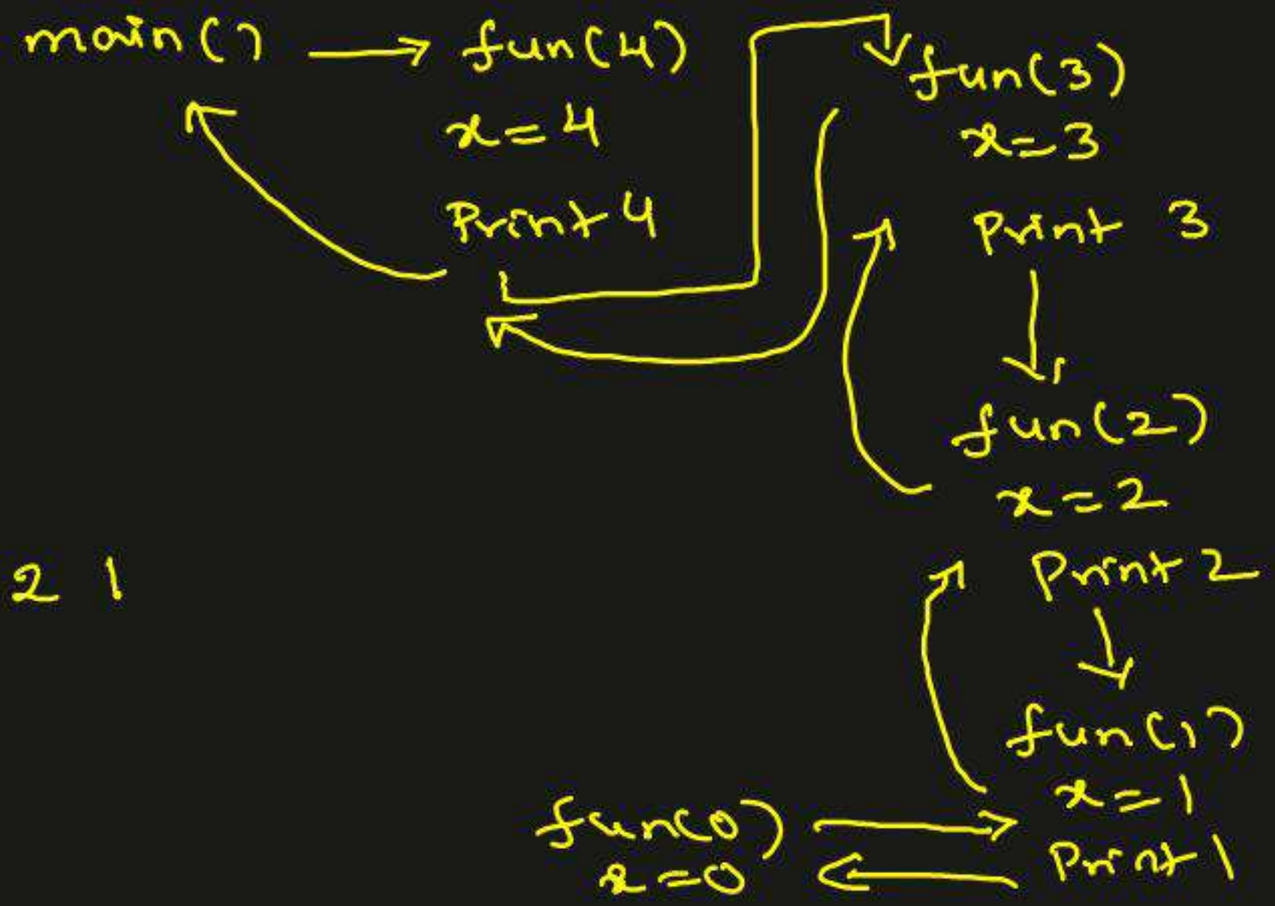
When the recursive call is the last or later statement in recursive function, then it is called Tail recursion.

Example:

```
void
~~int~~ fun(int x)
{
if(x==0) return;
else{
printf("x=%d",x);
fun(x-1);
}
}
```

```
void main( )
{ fun(4);
}
```

o/p: 4 3 2 1



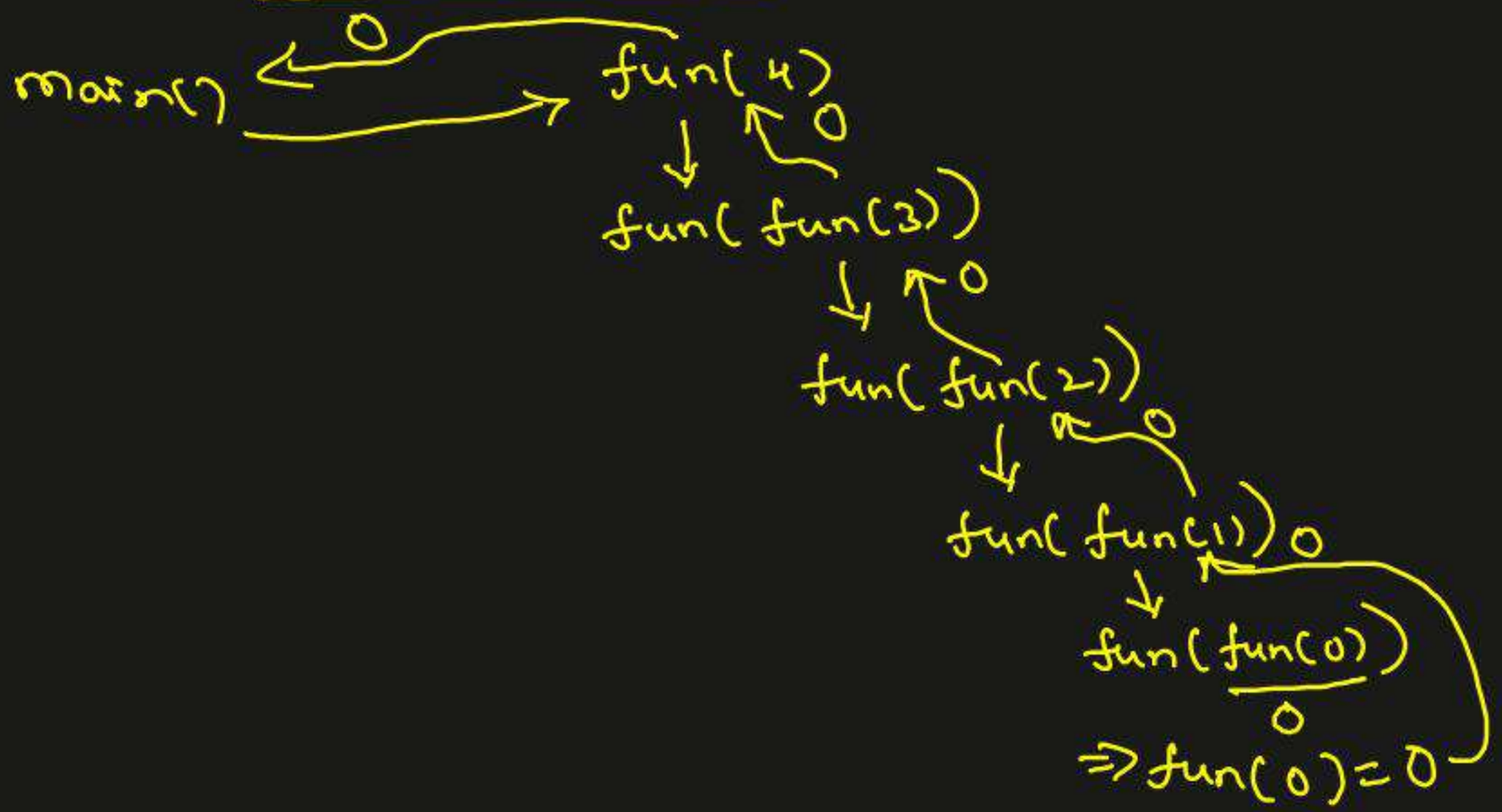
BYJU'S EXAM PREP

Nested Recursion

When the recursive call is made inside another recursive call and so on, then it is said to be nested recursion.

Example:

```
int fun(int x)
{
if(x==0) return; x;
else{
return fun(fun(x-1));
}
}
```



BYJU'S EXAM PREP

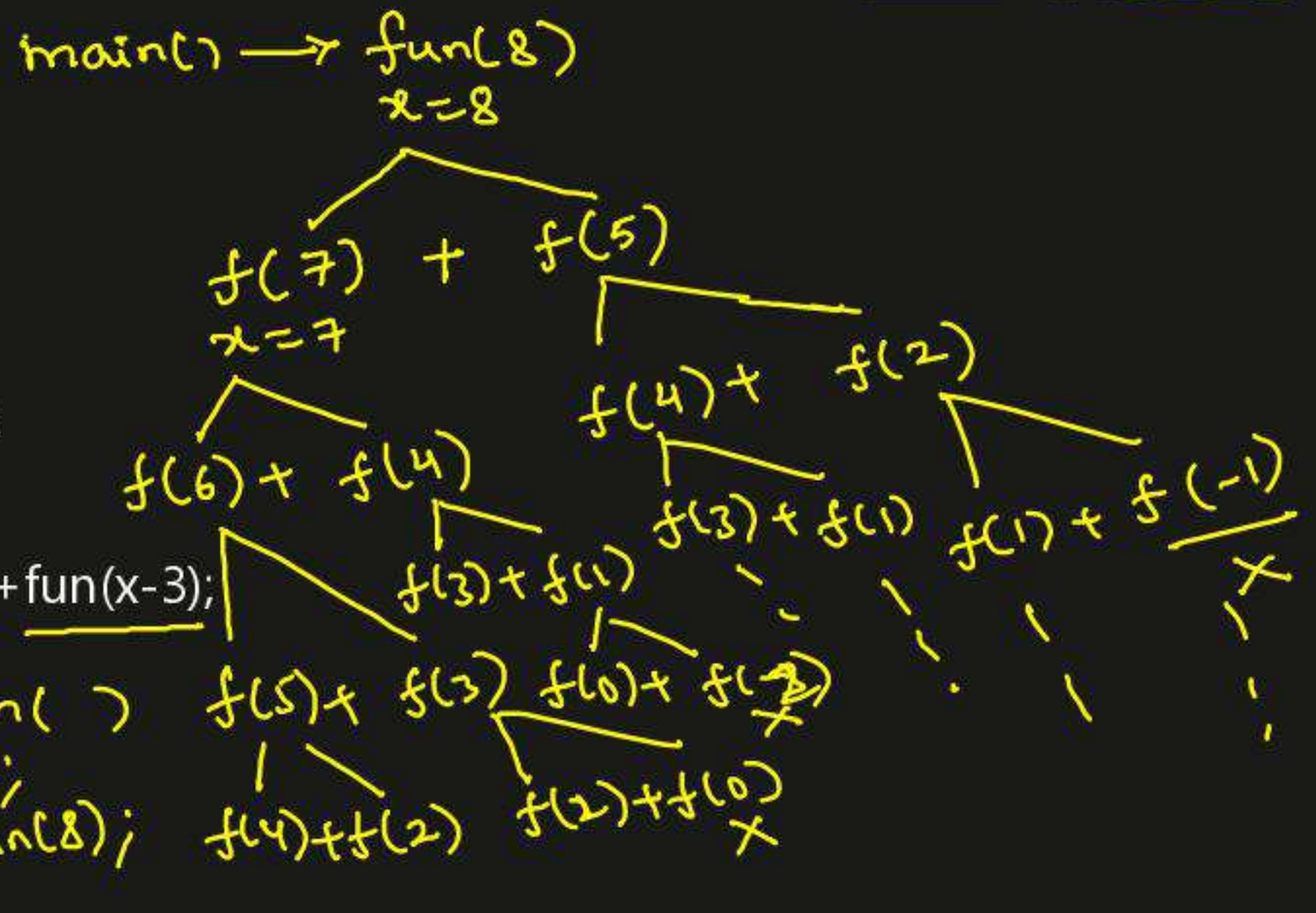
Tree Recursion

When the recursive call is made multiple times within the same function, then it is said to be Tree recursion.

Example:

```
int fun(int x)
{
if(x<=0) return;
else{
return fun(x-1)+fun(x-3);
}
}
```

```
Void main( )
{ int i;
  i=fun(8);
}
```



BYJU'S EXAM PREP

Indirect Recursion

by Multiple functions, Mutually, Then it is indirect Recursion

When the recursive call is made multiple times within the same function, then it is said to be Tree recursion.

```
int first_function(int num)
{
    if (num<=2)
        return 2;
    else
        return new_function(num);
}
int new_function(int num)
{
    return first_function(num-1);
}
```

first_func (5) 2
num=5
↳ new_fun(5) num=5
↓
first_fun(4)
↳ new(4)
↓
first(3)
↓
new(3) 2
↓ first(2)

BYJU'S EXAM PREP

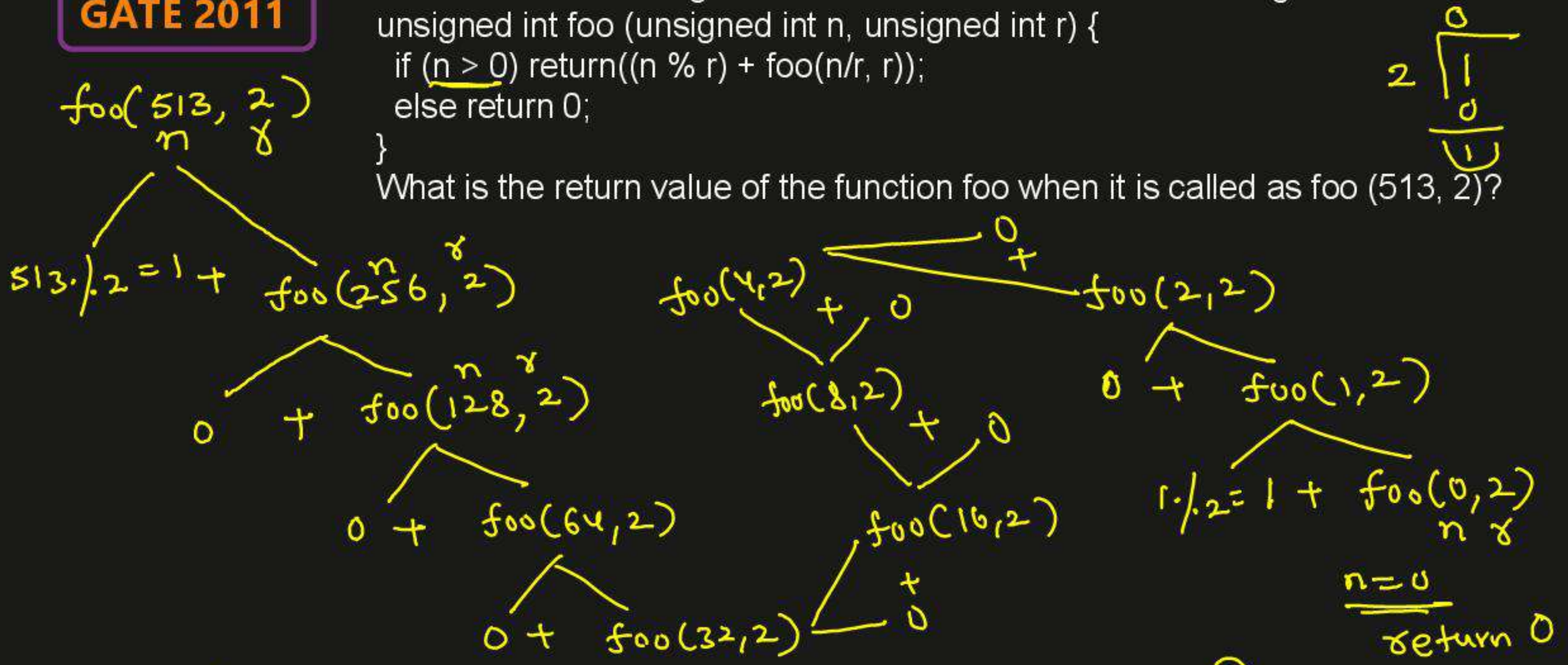
Ans: 2

GATE 2011

Consider the following recursive C function that takes two arguments:

```
unsigned int foo (unsigned int n, unsigned int r) {
  if (n > 0) return((n % r) + foo(n/r, r));
  else return 0;
}
```

What is the return value of the function foo when it is called as foo (513, 2)?



$1+0+0+0+0+0+0+0+0+1+0 == 2$

BYJU'S EXAM PREP

GATE 1998

What value would the following function return for the input x = 95?

```
Function fun (x:integer):integer;
Begin
    If x > 100 then fun : x – 10
    Else fun : fun(fun (x + 11))
End;
```

Ans: 91

fun(95)

fun(fun(106))

96

fun(96)

↳ fun(fun(107))

x>100 ⇒ 97

fun(97)

↳ fun(fun(108))

98

fun(98)

↳ fun(fun(109))

99

fun(99)

↳ fun(fun(110))

100

fun(100)

↳ fun(fun(111))

101

fun(101)

x>100

= 101 − 10

= 91

Pointers & Arrays - 1

GATE 2018

Counter = ~~0~~ ~~7~~ ~~3~~ 4

Consider the following C program:

```
#include < stdio.h >
  int counter = 0;
  int calc (int a, int b) {
  int c;
  counter++;
  if (b==3) return (a*a*a);
  else {
    c = calc(a, b/3);
    return (c*c*c);
  }
}
int main (){
  calc(4, 81);
  printf ("%d", counter);
}
```

The output of this program is _____.

Variable declared outside all functions ⇒ Global variable/ External variable.
⇒ It can be used throughout program.

main → calc(4, 81) (a, b)
counter++;
c = calc(4, 27)
c = 3

calc(4, 27) (a, b)
counter++;
c = calc(4, 9)
c = 3

calc(4, 9) (a, b)
counter++
c = calc(4, 3)
c = 3

calc(4, 3) (a, b)
counter++
(4*4*4)

Counter = 4

BYJU'S EXAM PREP

Pointers & Arrays - 1

BYJU'S EXAM PREP

Homework

GATE 2005

Consider the following C program:

```
void foo (int n, int sum) {
  int k = 0, j = 0;
  if(n==0) return;
  k=n%10; j = n /10;
  sum = sum + k;
  foo(j, sum);
  printf("%d",k);
}
int main () {
  int a = 2048, sum = 0;
  foo(a, sum);
  printf("%d\n", sum);
}
```

What does the above program print?

BYJU'S EXAM PREP

Yesterday's Home work

Q4. Consider the following C program

```
main()
{
  int x, y, m, n;
  scanf ("%d %d", &x, &y);
  /* x > 0 and y > 0 */
  m = x; n = y;
  while (m != n)
  {
    if(m>n)
      m = m - n;
    else
      n = n - m;
  }
  printf("%d", n);
}
```

Let $x = 36$ $y = 24$

$m = 36$ $n = 24$

$36 != 24$ (True)

$m = 12$

$12 != 24$ (True)

$n = 12$

$\hookrightarrow 12 != 12$ False

Print 12

$m = 5$ $n = 7$

$5 != 7$ (T)

$5 > 7$ (F) $n = 2$

$7 != 7$ (T)

$m = 3$

The program computes

[GATE 2004]

A. x + y using repeated subtraction. ($36 + 24 = 60$)

B. x mod y using repeated subtraction. ($36 \% 24 = 1$)

C. The greatest common divisor of x & y. ($gcd(36, 24) = 12$) ✓

D. The least common multiple of x & y. ($LCM(36, 24) = 72$)

$x=5 \quad y=7$

$m=5 \quad n=7$

$5 \,\%\, 7 = 7 \; (T) \qquad 5 > 7 \; (F) \qquad n = n-m$

$n = 2$

$m=5 \quad n=2$

$5 \,\%\, 2 = 2 \; (T) \qquad 5 > 2 \; (T) \qquad m = m-n$

$m = 3$

$m=3 \quad n=2$

$3 \,\%\, 2 = 2 \; (T) \qquad 3 > 2 \; (T) \qquad m = 3-2$

$= 1$

$m=1 \quad n=2$

$1 \,\%\, 2 = 2 \; (T) \qquad 1 > 2 \; (F) \qquad n = n-m$

$n = 2-1 = 1$

$m=1 \quad n=1$

$1 \,\%\, 1 = 1 \; (F)$

Print n = Print '1'

$x+y = 5+7 = 12$

$x \cdot y = 5 \cdot 7 = 5$

$gcd(5,7) = 1$

$LCM(5,7) = 35$

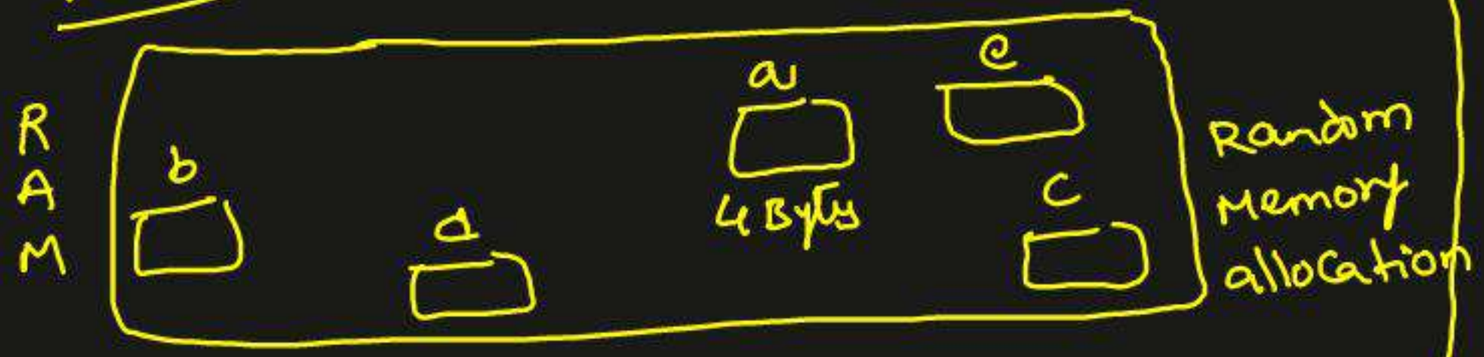
BYJU'S EXAM PREP

Array

- An array is group of similar or homogeneous data items.
  - Integers group
  - characters group
  - float values group
- All elements of array store at consecutive locations in memory.
- Arrays support random access with index. (By default, Index starts with '0')
- There are 2 types of arrays : one-dimensional and multi-dimensional [2D, 3D, 4D, - - n-D]

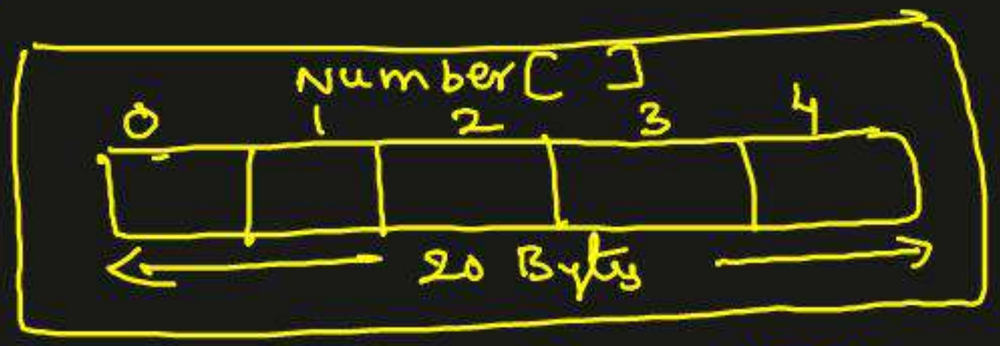
Let, 32-bit Processor

int a, b, c, d, e;



⟹

int Number[5];



Contiguous Memory space.

BYJU'S EXAM PREP

1-D Array Declaration

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

- type arrayName [arraySize];
- This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type.
- Example: double balance[5];

int x[7];

char str[10];

int x[7.43];

Error, Size must be integer

BYJU'S EXAM PREP

1-D Array Initialization

- You can initialize an array in C either one by one or using a single statement as follows –
- Example: double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].
- If you omit the size of the array, based on values assigned, size is considered by complier.
- Example: double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

BYJU'S EXAM PREP

Accessing Array Elements

```
int x[5], i;
for(i=0; i<5; i++)
  scanf("%d", &x[i]);
```

i = 0	1	2	3	4
10	20	30	40	50

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

```
for(i=0; i<5; i++)
  Printf("%d", x[i]);
```

- For example – double salary = balance[9];

The above statement will take the 10th element from the array and assign the value to salary variable.

```
int x[5];  Printf("%d", x[3]);  // Garbage value
```

```
int x[5]= {10, 20};
Printf("%d", x[3]);  =0
```

0	1	2	3	4
10	20	0	0	0

If atleast one value is initialized, Then zeroes will be other values.

Address = %u

"u" means unsigned integer

Pointers and 1-D Arrays

Let 32-bit Processor

Let, int a [5] = {10, 20,30,40,50};

20 Bytes

Elements	a[0]	a[1]	a[2]	a[3]	a[4]
Value	10	20	30	40	50
Address	1000	1004	1008	1012	1016

Base address

a= &a[0]=1000

20 Bytes

Group :-

1000

1019 addresses

BYJU'S EXAM PREP

Pointers and 1-D Arrays

array name itself means base address.

If 'p' is declared as integer pointer, then, an array 'a' can be pointed by the following assignment

int *p;

p = a; (or) p = &a[0];

Printf("%d", a); ⇒ 1000

Print p ⇒ 1000

Every value of 'a' is accessed by using p++ to move from one element to another element. When a pointer is incremented, its value is increased by the size of the datatype that it points to. This length is called the "scale factor".

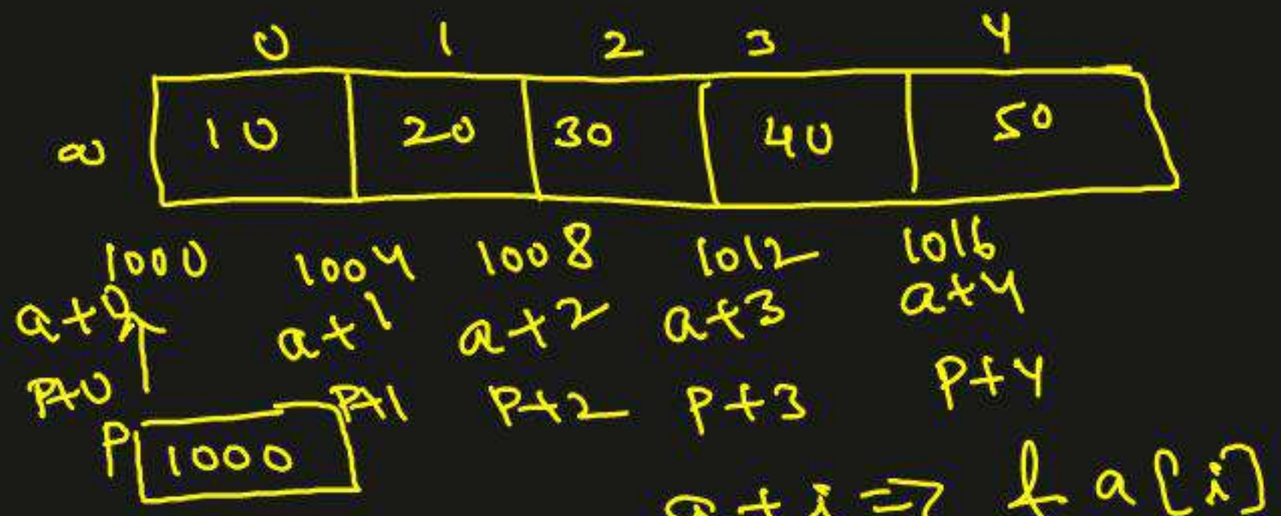
p = &a[0] = 1000 = a+0

p+1 = &a[1] = 1004 = a+1

p+2 = &a[2] = 1008 = a+2

p+3 = &a[3] = 1012 = a+3

p+4 = &a[4] = 1016 = a+4



a+i ⇒ &a[i]

BYJU'S EXAM PREP

Pointers and 1-D Arrays

a == p

1-D array is Equivalent to Pointers.

Address of an element is calculated using its index and the scale factor of the datatype. Address of a[3] = base address + (3* scale factor of int)

= 1000 + (3*4)

= 1000 +12

= 1012

& a[i] = Base address + i*size

& a[3] = 1000 + 3*4

= 1012

Pointers can be used to access array elements instead of using array indexing.

*(p+3) gives the value of a[3].

*(p+3) ⇒ value at address (p+3)

⇒ value at address (1012)

= 40

= a[3]

a[i] = *(p+i)

BYJU'S EXAM PREP

Examples

Pointers & Arrays - 1

To be contd...

Q1. Consider the following C program :

```
#include < stdio.h >

 int main ()  {

        int arr [ ] = {1,2,3,4,5,6,7,8,9,0,1,2,5}, *ip = arr+4;

         printf ("%d\n", ip[1]);

         return 0;

}
```

The number that will be displayed on execution of the program is _____.

[GATE 2019]

BYJU'S EXAM PREP

Q2. Consider the following C program :

```
#include < stdio.h >
int main()
{
    int a[] = {2, 4, 6, 8, 10} ;
    int i, sum = 0, *b = a + 4 ;
    for (i = 0; i < 5; i++)
        sum = sum + (*b - i) - *(b - i) ;
    printf ("%d\n", sum) ;
    return 0 ;
}
```

The output of the above C program is ____.

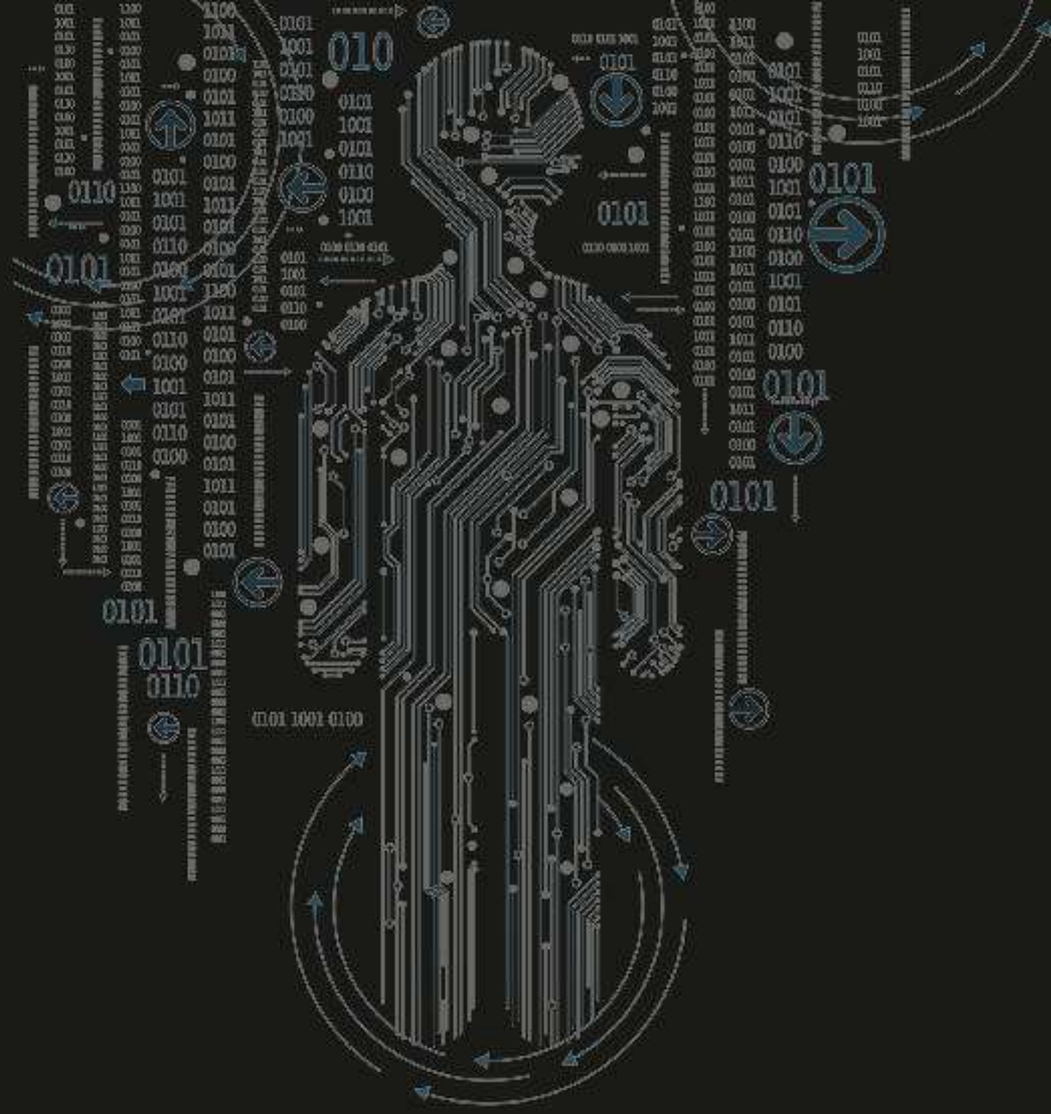
[GATE 2019]

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Pointers & Arrays - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

$n = 9$

```
int a=1
for( i=1; i<=n; i++)
{ for(j=1; j<=i; j++)
      for(k=1; k<=j; k++)
          a=a+1;
}
Pf("%d", a);
```

Exp2 Exp2 Exp2 Exp3

$i=1$ $1<=9$ T

$j=1$ $1<=1$ T

$k=1$ $k<=1$ T

$a=a+1 \Rightarrow a=2$

$k=2$ $2<=1$ F

$j=2$ $2<=1$ F

$i=2$ $2<=9$ True

$j=1$ $1<=2$

$k=1$ $1<=1$ T $a=3$

$k=2$ False

$j=2$ $2<=2$

$k=1$ $1<=2$ True

$a=4$

$k=2$ $a=5$

$a=1$ $\quad i=1 \quad i=2 \quad i=3 \quad i=4 \quad i=5$

$a+1+(1+2)+(1+2+3)+(1+2+3+4)+(1+2+3+4+5)$

$i=6 \quad i=7$

$(1+2+3+4+5+6)+(1+2+3+4+5+6+7)$

$i=3 \quad 3<=9$

$j=1$

$k=1 \quad a=6$

$j=2$

$k=1$, $k=2$ } $a=\cancel{7}\,8$

$j=3$

$k=1$, $k=2$, $k=3$ } $a=\cancel{9}\,10\,11$

$+ \quad i=8 \quad i=9$

$(1+2+3+4+5+6+7+8)+(1+2+3+4+5+6+7+8+9)$

$a=1+1+3+6+10+15+21+28+36+45=$

$a = 10$

$a, b = (b, a)$

$a, b = a$

$a = 10, b = 10$

$a, b = b, \cancel{a};$

$a, b = (b, a);$

Pointers & Arrays - 2

BYJU'S EXAM PREP

Last Class – Quick Revision

	0	1	2	3
a	10	20	30	40
	1000	1004	1008	1012
	p	p+1	p+2	p+3

a[0] = 0[a]

a[3] / 3[a]

Array – Collection of similar data
- 1-D
- Multi-D [2D, 3D, 4D ---]

$\boxed{a[i] = i[a]}$

Pointers & 1-D arrays

int Name[size] = {values};

- Name[index] = value;
- Atleast one value is initialized, then other initialized values will be zeroes.
- If None is initialized, then garbage values.

p = &a; Let array = a, pointer = p

a = address

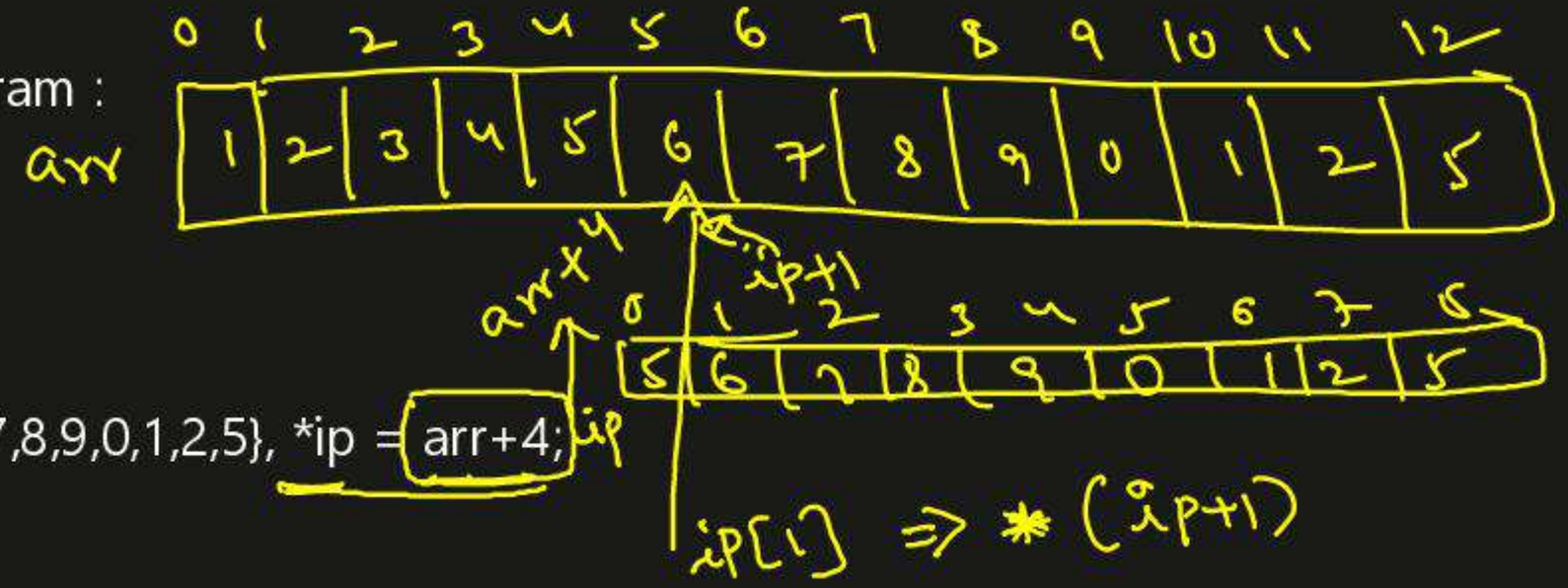
a + i ⇒ address of a[i] == p+i

a[i] == p[i]

Pointers & Arrays - 2

Q1. Consider the following C program :

```
#include < stdio.h >

 int main () {

        int arr [ ] = {1,2,3,4,5,6,7,8,9,0,1,2,5}, *ip = arr+4;

         printf ("%d\n", ip[1]);

         return 0;

}
```



The number that will be displayed on execution of the program is _____.

Ans:- 6

[GATE 2019]

Pointers & Arrays - 2

Q2. Consider the following C program :

```
#include < stdio.h >
int main()
{
    int a[] = {2, 4, 6, 8, 10} ;
    int i, sum = 0, *b = a + 4 ;
    for (i = 0; i < 5; i++)
        sum = sum + (*b - i) - *(b - i) ;
    printf ("%d\n", sum) ;
    return 0 ;
}
```

Sum = ~~0~~ p ~~y~~ ~~3~~ ~~4~~ 10

ANS: 10

	0	1	2	3	4
a	2	4	6	8	10
	b-4	b-3	b-2	b-1	b

i=0 Sum = 0 + (10-0) - 10 = 0+10-10 = 0

i=1 Sum = 0 + (10-1) - 8 = 9-8 = 1

i=2 Sum = 1 + (10-2) - 6 = 3

i=3 Sum = 3 + (10-3) - 4 = 6

i=4 Sum = 6 + (10-4) - 2 = 6+6-2 = 10

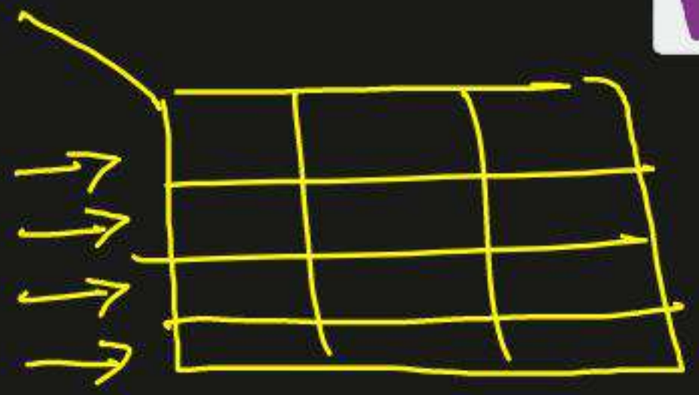
The output of the above C program is ____.

[GATE 2019]

BYJU'S EXAM PREP

2-D Array

1-D array : Vector.



- A two-dimensional array is also known as matrix.
- Syntax for Declaration: datatype name [rows] [cols];
- A 2-D array may be stored in memory in either of 2 ways:
- Row – Major Order (RMO) C, C++, Java
- Column- Major Order (CMO) Fortran, MATLAB, Scilab, R languages.

int A[3][4] = {10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65};

	col 0	col 1	col 2	col 3
row 0	10	15	20	25
row 1	30	35	40	45
row 2	50	55	60	65

Let 32-bit Processor

⇒ 1 int = 4 Bytes

RMO

48 Bytes

1000 1004 1008 1012 1016

RMO:

10	15	20	25	30	35	40	45	50	55	60	65

1000 row 0 row 1 row 2 1044 1047

CMO:

10	30	50	15	35	55	20	40	60	25	45	65

col 0 col 1 col 2 col 3

1008

Nesting of loops

RMO:

```
for(i=0; i<r; i++)
{
    for(j=0; j<c; j++)
    {
        scanf("%d", &A[i][j]);
    }
}
```

i=0	A[i][j]
j=0	A[0][0]
j=1	A[0][1]
j=2	A[0][2]
⋮	

i=1	
j=0	A[1][0]
j=1	A[1][1]
j=2	A[1][2]
⋮	

CMO:

```
for(j=0; j<c; j++)
{
    for(i=0; i<r; i++)
        A[i][j]
```

j=0	i j
i=0	A[0][0]
i=1	A[1][0]
i=2	A[2][0]
⋮	

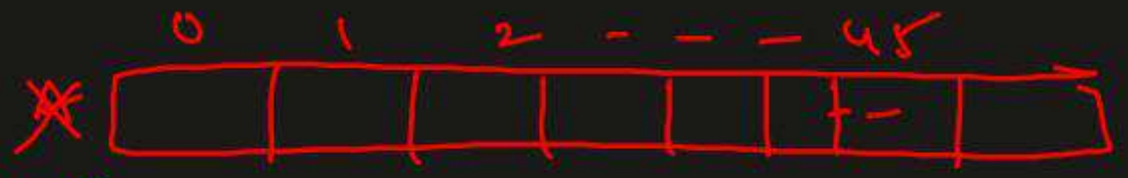
j=1	
i=0	A[0][1]
i=1	A[1][1]
i=2	A[2][1]
⋮	

Pointers & Arrays - 2

BYJU'S EXAM PREP

RMO, CMO

Let int, 4 bytes



&x[45] ⇒ 1000 + 45*4 = 1000 + 180 = 1180

- 1-D Array :

  &X[i] = B + i * n Index from 0

  &X[i]= B + (i-1) * n Index From 1

B = Base address
i = index
n = size of each element.

- 2-D Array :

  Case 1 : If Index Starts from '0'

  &X[i][j]= B +[(i*c) + j] * n Row Major Order

  &X[i][j]= B +[(j*r) + i] * n Column Major Order

  Case 2 : If Index Starts from '1'

  &X[i][j]= B +[(i-1*c) + j-1] * n Row Major Order

  &X[i][j]= B +[(j-1*r) + i-1] * n Column Major Order

r = Total no. of rows
c = Total no. of cols
i = row index
j = col index

BYJU'S EXAM PREP

Address Of an Element

Let 'A' be Array and 'P' be pointer and 'i' be index

- For 1-D array :

  A[i] == P[i] == i[A] == i[P] == Value

  A+i == i+A == P+i == i+P == Address

- For 2-D array :

  A + i == * (A + i) == & A[i][] == Address of ith row

  *(A + i) + j == Address of ith row and jth column

  ((A + i) + j) == A[i][j] == Value A[i][j]

int **p; p = &A;

A

	0	1	2	3
A+0 → Base address of 0th row	1000 10	1004 20	1008 30	1012 40
A+1 → Base address of 1st row	1016 50	1020 60	1024 70	1028 80
A+2 →	1032 90	1036 5	1040 15	1044 25

A+i == Address of ith row == *(A+i)

*(A+i)+j = Address of ith row, jth col

((A+i)+j) = A[i][j]

*(A+0) = Value at address (A+0)

A+0 gives 0th row address

*(A+0)+2 = 1000 + 2*4

= 1008

BYJU'S EXAM PREP

Examples

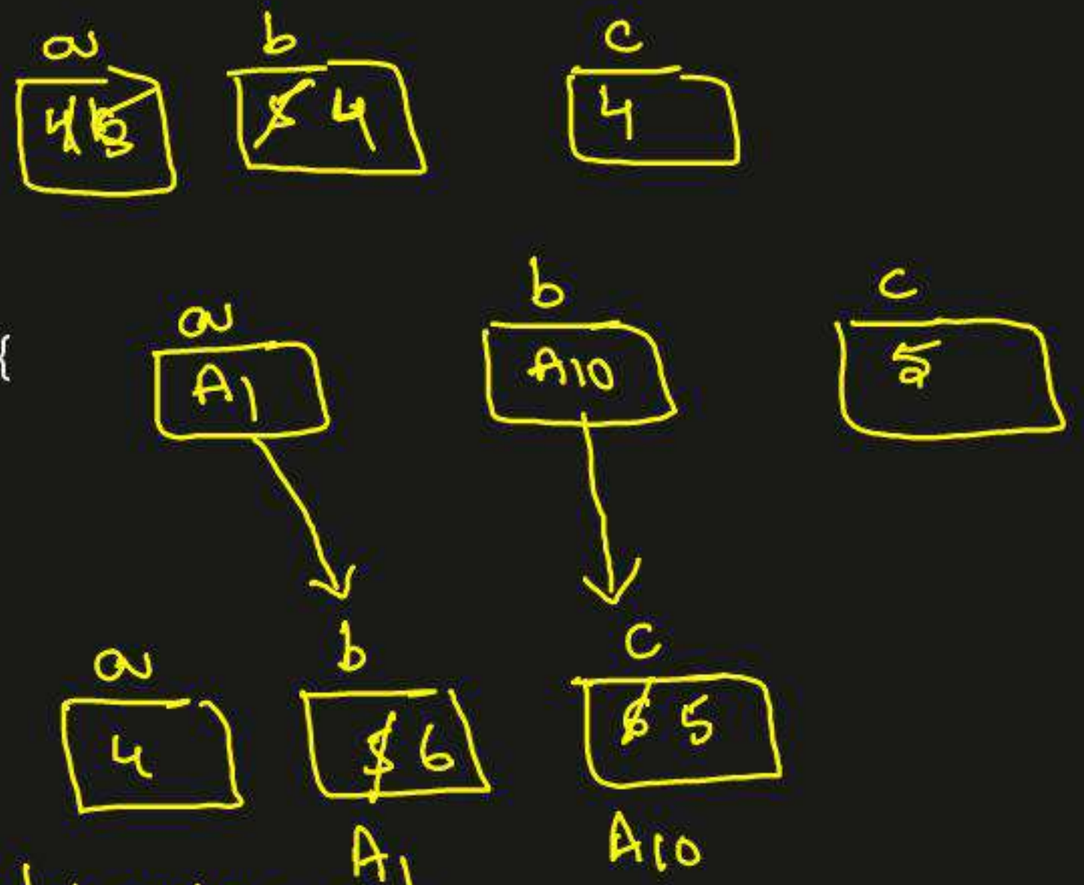
Pointers & Arrays - 2

BYJU'S EXAM PREP

Q1. The output of the following C program is_____. [GATE 2015]

```
void f1(int a, int b) {
int c;
c=a; a=b; b=c;
}
void f2(int *a, int *b) {
int c;
c=*a; *a=*b; *b=c;
}
int main(){
int a=4, b=5, c=6;
f1(a,b);
f2(&b, &c);
printf("%d",c-a-b);
}
```

← Call-by-reference



Ans: −5

$c - a - b = 5 - 4 - 6$

$= 5 - 10 = -5$

Pointers & Arrays - 2

Q2. Consider the following C program. Home-work [GATE 2016]

```
#include < stdio.h >
    void mystery(int *ptra, int *ptrb) {
    int *temp;
    temp = ptrb;
    ptrb = ptra;
    ptra = temp;
}
int main() {
    int a=2016, b=0, c=4, d=42;
    mystery(&a, &b);
    if (a < c)
        mystery(&c, &a);
    mystery(&a, &d);
    printf("%d\n", a);
}
```

The output of the program is _____.

Pointers & Arrays - 2

Home-work

Q3. Consider the following C program:

```
#include <stdio.h>
void fun1(char *s1, char *s2) {
  char *temp;
  temp = s1;
  s1 = s2;
  s2 = temp;
}
void fun2(char **s1, char **s2) {
  char *temp;
  temp = *s1;
  *s1 = *s2;
  *s2 = temp;
}
int main() {
  char *str1 = "Hi", *str2 = "Bye";
  fun1(str1, str2);
  printf("%s %s", str1, str2);
  fun2(&str1, &str2);
  printf("%s %s", str1, str2);
  return 0;
}
```

The output of the program above is [GATE 2018]

A. Hi Bye Bye Hi

B. Hi Bye Hi Bye

C. Bye Hi Hi Bye

D. Bye Hi Bye Hi

Pointers & Arrays - 2

Home work

Q4. What is printed by the following C program?

```
#include <stdio.h>
int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
   z  = **ppz;
  *py += 2;
   y = *py;
   x += 3;
   return x + y + z;
}
```

```
void main()
{
   int c, *b, **a;
   c = 4;
   b = &c;
   a = &b;
   printf( "%d", f(c,b,a));
   getchar();
}
```

[GATE 2008]

A. 18

B. 19

C. 21

D. 22

Pointers & Arrays - 2

BYJU'S EXAM PREP

$*(x+2)+3 = 2024 + 3(4)$
$= 2036$

Q5. What is the output of the following C code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory.

```
#include <stdio.h>
int main()
{
   unsigned int x[4][3] = {{1, 2, 3}, {4, 5, 6},
                           {7, 8, 9}, {10, 11, 12}};
   printf("%u, %u, %u", x+3, *(x+3), *(x+2)+3);
}
```

2036 2036 2036.

	0	1	2
0	2000 1	2004 2	2008 3
1	2012 4	2016 5	2020 6
2	2024 7	2028 8	2032 9
3	2036 10	2040 11	2044 12

[GATE 2015]

A. 2036, 2036, 2036

B. 2012, 4, 2204

C. 2036, 10, 10

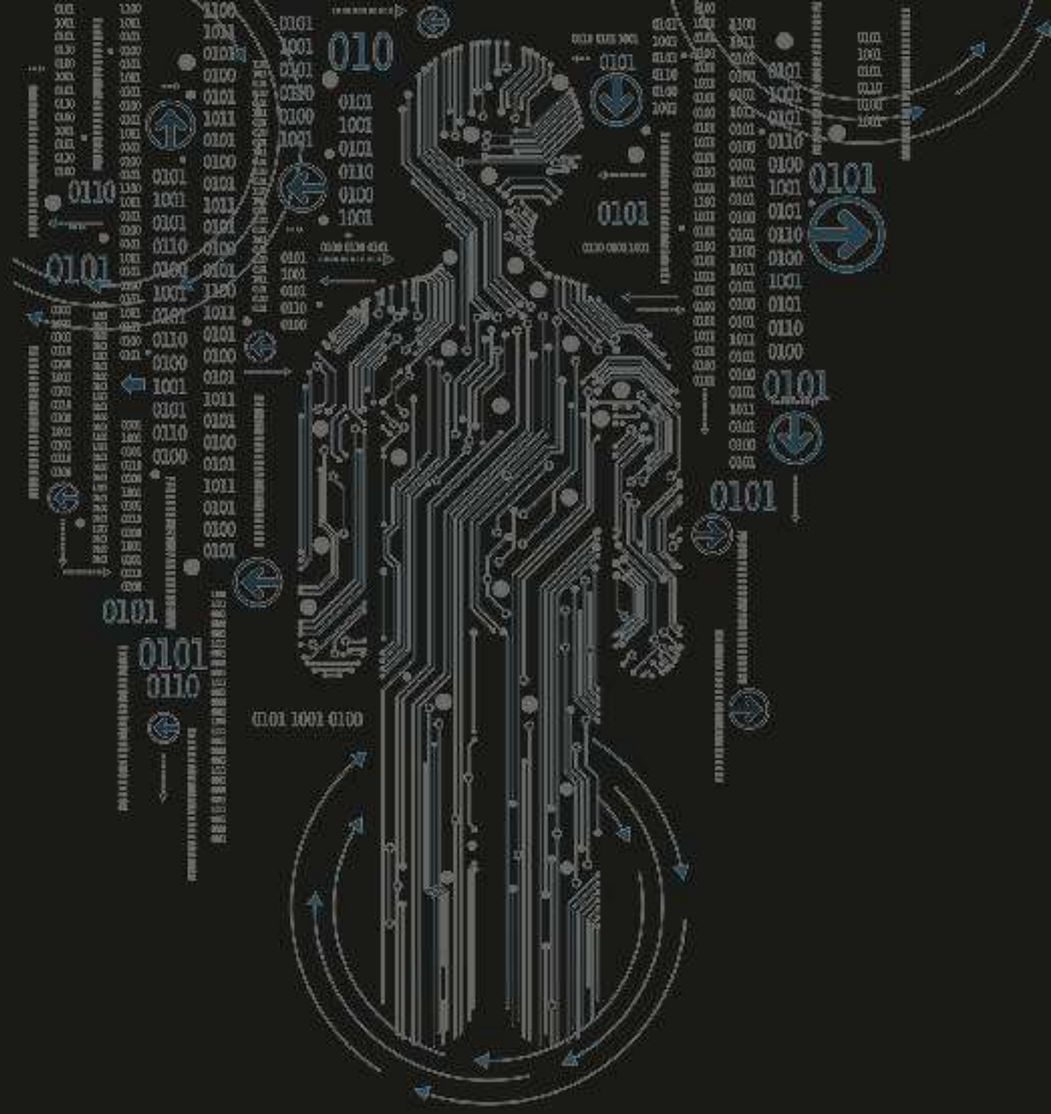
D. 2012, 4, 6

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Pointers & Arrays - 3



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Pointers & 2-D arrays

"X" – 2D array, **p

- (x+i) == *(x+i) = & x[i][ ]
- *(x+i)+j = & x[i][j]
- *(*(x+i)+j) = x[i][j]

- Address of an element in 1-D array
  2-D array.

BYJU'S EXAM PREP

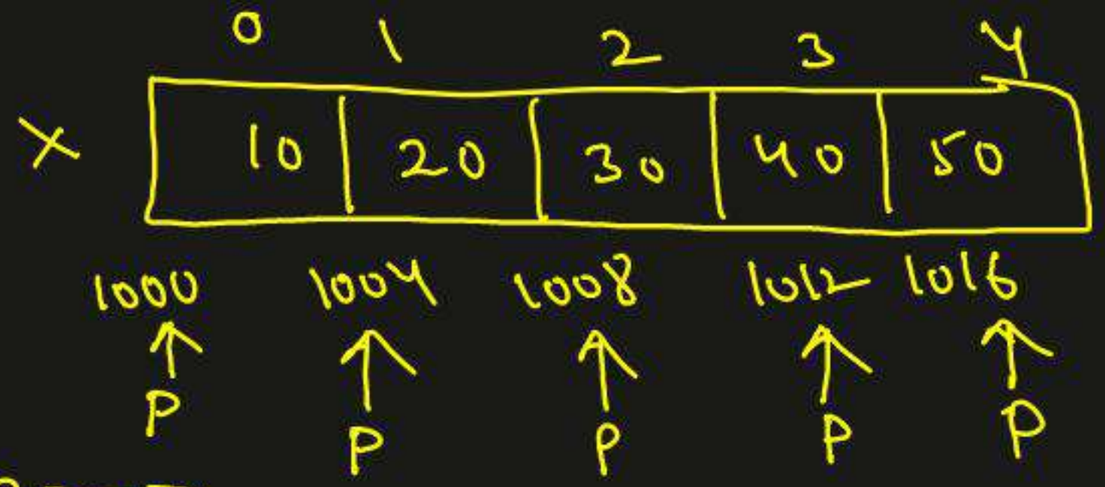
Pointers and Arrays

Arrays and Pointers can be associated together in 3 ways:

- A pointer to an element of an array : Pointer
- A pointer to Whole Array : Pointer array
- Array of Pointers :

BYJU'S EXAM PREP

A pointer to an element of an Array




```
int x[5]={10,20,30,40,50}, *p;

p= &x[0];
```

P 1000

P= &x[2] P 1008 P=x+4; P 1016

In the above code, pointer 'p' points to a single element of an array only.

Let P= &x[0]; ⇒ P ~~1000~~ ~~1004~~ 1012

P++; ⇒ 1004

P+2 ⇒ 1004 + 2(4) = 1004+8 = 1012

BYJU'S EXAM PREP

A pointer to Whole Array

: Pointer to array / array Pointer

When a pointer is created to store the address of an array, such pointer is called a pointer to an array. Pointer to an array is also known as an array pointer.

Syntax: data_type (*var_name)[size_of_array];

Example: int (*ptr)[5];

Here ptr is a pointer that can point to an array of 5 integers. Since subscript has higher precedence than indirection, it is necessary to enclose the indirection operator and pointer name inside parentheses. Here the type of ptr is 'pointer to an array of 5 integers'.

Let x[5] = {10, 20, 30, 40, 50};
int (*p)[5];
p = &x;

Access
(*p)[0]
(*p)[1]

Pt+; ⇒ 1020
P+2 ⇒ 1020 + 2(20)
= 1060

	0	1	2	3	4
x	10	20	30	40	50
Address:	1000	1004	1008	1012	1016

P [1000]

BYJU'S EXAM PREP

Array Of pointers

It is basically an array of the pointer variables. ~~It is also known as pointer arrays~~.

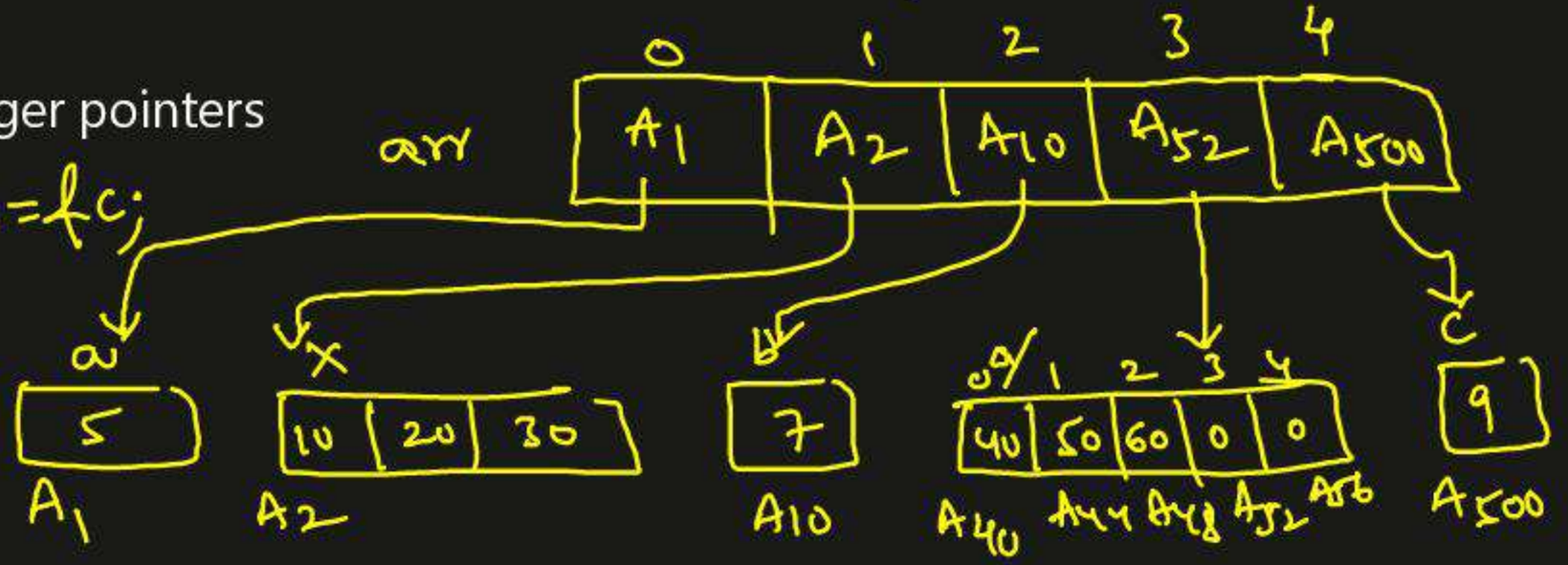
Declaration of an array of pointers:

Syntax: data_type *arrar_name[array_size];

Example: int *arr[5];

Here "arr" is an array of 5 integer pointers

```
int a=5, x[3]={10,20,30};
int b=7, y[5]={40,50,60};
int c=9;
```




```
arr[0]= &a;        arr[4]=&c;
arr[1]= &x[0] or &x;
arr[2]= &b;
arr[3]= &y[3];
```

BYJU'S EXAM PREP

Strings in 'C'

```
char str[5] = "Hello";
Printf("%s", str);
```

o/p
Hell

	0	1	2	3	4
str	H	e	l	l	\0

In C programming, a string is a sequence of characters terminated with a null character '\0'.

For example: char c[] = "c string";

```
Char str[ ] = "Hello";
Print str ⟶ Hello
```

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default as shown below:

H	e	l	l	o	\0

c		s	t	r	i	n	g	\0

BYJU'S EXAM PREP

Strings Declaration and Initialization

int array
atleast one value, remaining : 0
float array
atleast one value, remaining: 0.000000

In C programming, a string is declared as character array: char s[5];

You can initialize strings in a number of ways.

char : '\0'

if None is initialized: Garbage values

- char c[] = "abcd";
- char c[50] = "abcd";
- char c[] = {'a', 'b', 'c', 'd', '\0'};
- char c[5] = {'a', 'b', 'c', 'd', '\0'};

optional

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

BYJU'S EXAM PREP

char c[100];
char str[20]="c Programming";
c=str; //valid

Reading a String

Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared.

For example, char c[100]; = "c Programming"; //valid

c = "C programming"; // Error! array type is not assignable.

strcpy(c, str);

Use the strcpy() function to copy the string instead or scanf() function to read a string. The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

You can use the gets() function to read a line of string. And, you can use puts() to display the string.

BYJU'S EXAM PREP

String Handling Functions

Commonly Used String Functions :

strlen() - Calculates the length of a string

strcpy() - Copies a string to another

strcmp() - Compares two strings

strcat() - Concatenates two strings

strlwr() – Converts string to lower case

strupr() – Converts string to upper case

Pointers & Arrays - 3

BYJU'S EXAM PREP

Q1. What does the following fragment of C-program print?

```
char c[] = "GATE2011";
char *p = c;
printf("%s", p + p[3] - p[1]) ;
```

%d

p+4

%c ⇒ 2

	0	1	2	3	4	5	6	7
c	G	A	T	E	2	0	1	1

p

p+4

A = 65
B = 66
C = 67
D = 68
E = 69

[GATE 2011]

A. GATE2011

B. E2011

C. 2011 ✓

D. 011

p + 'E' − 'A'

p + 69 − 65

p + 4

NOTE:- NULL is Never printed.

Pointers & Arrays - 3

BYJU'S EXAM PREP

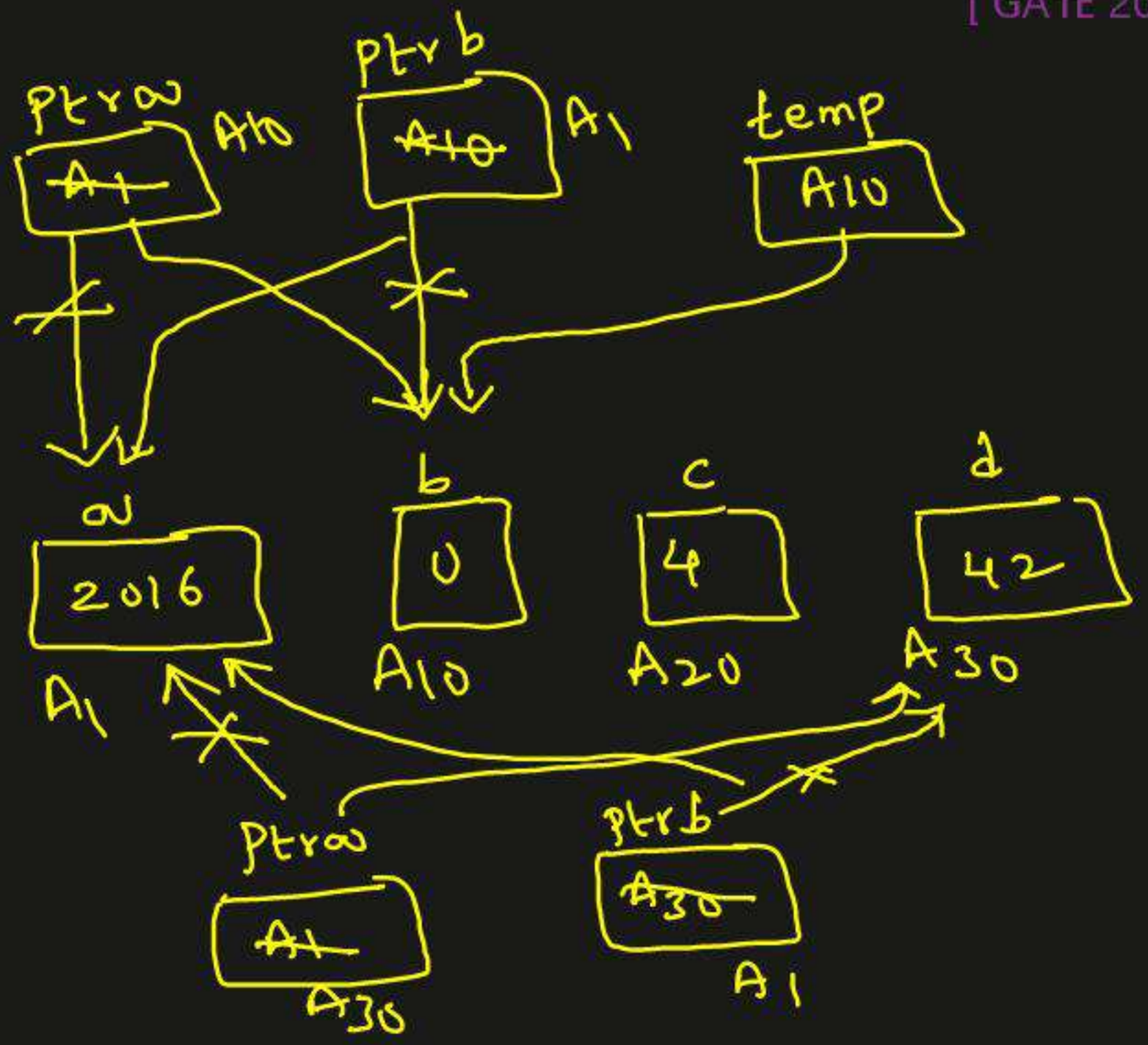
Q2. Consider the following C program. [GATE 2016]

```
#include < stdio.h >
    void mystery(int *ptra, int *ptrb) {
    int *temp;
    temp = ptrb;
    ptrb = ptra;
    ptra = temp;
}
int main() {
    int a=2016, b=0, c=4, d=42;
    mystery(&a, &b);
    if (a < c)
        mystery(&c, &a);
    mystery(&a, &d);
    printf("%d\n", a);
}
```

False

ANS: 2016

The output of the program is _____.



Pointers & Arrays - 3

Str1 | H | i | \0 | A1 Bye

Str2 | B | y | e | \0 | A10 Hi

Q3. Consider the following C program:

```
#include <stdio.h>
void fun1(char *s1, char *s2) {
  char *temp;
  temp = s1;
  s1 = s2;
  s2 = temp;
}
void fun2(char **s1, char **s2) {
  char *temp;
  temp = *s1;
  *s1 = *s2;
  *s2 = temp;
}
```

s1: Hi, Bye s2: Bye, Hi

s1: A1 s2: A10

temp: Hi

```
int main() {
  char *str1 = "Hi", *str2 = "Bye";
  fun1(str1, str2);
  printf("%s %s", str1, str2);
  fun2(&str1, &str2);
  printf("%s %s", str1, str2);
  return 0;
}
```

Hi Bye

Bye Hi

The output of the program above is [GATE 2018]

A. Hi Bye Bye Hi

B. Hi Bye Hi Bye

C. Bye Hi Hi Bye

D. Bye Hi Bye Hi

Pointers & Arrays - 3

Home work

Q4. What is printed by the following C program?

```
#include <stdio.h>
int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
   z  = **ppz;
  *py += 2;
   y = *py;
   x += 3;
   return x + y + z;
}
```

```
void main()
{
   int c, *b, **a;
   c = 4;
   b = &c;
   a = &b;
   printf( "%d", f(c,b,a));
   getchar();
}
```

[GATE 2008]

A. 18

B. 19

C. 21

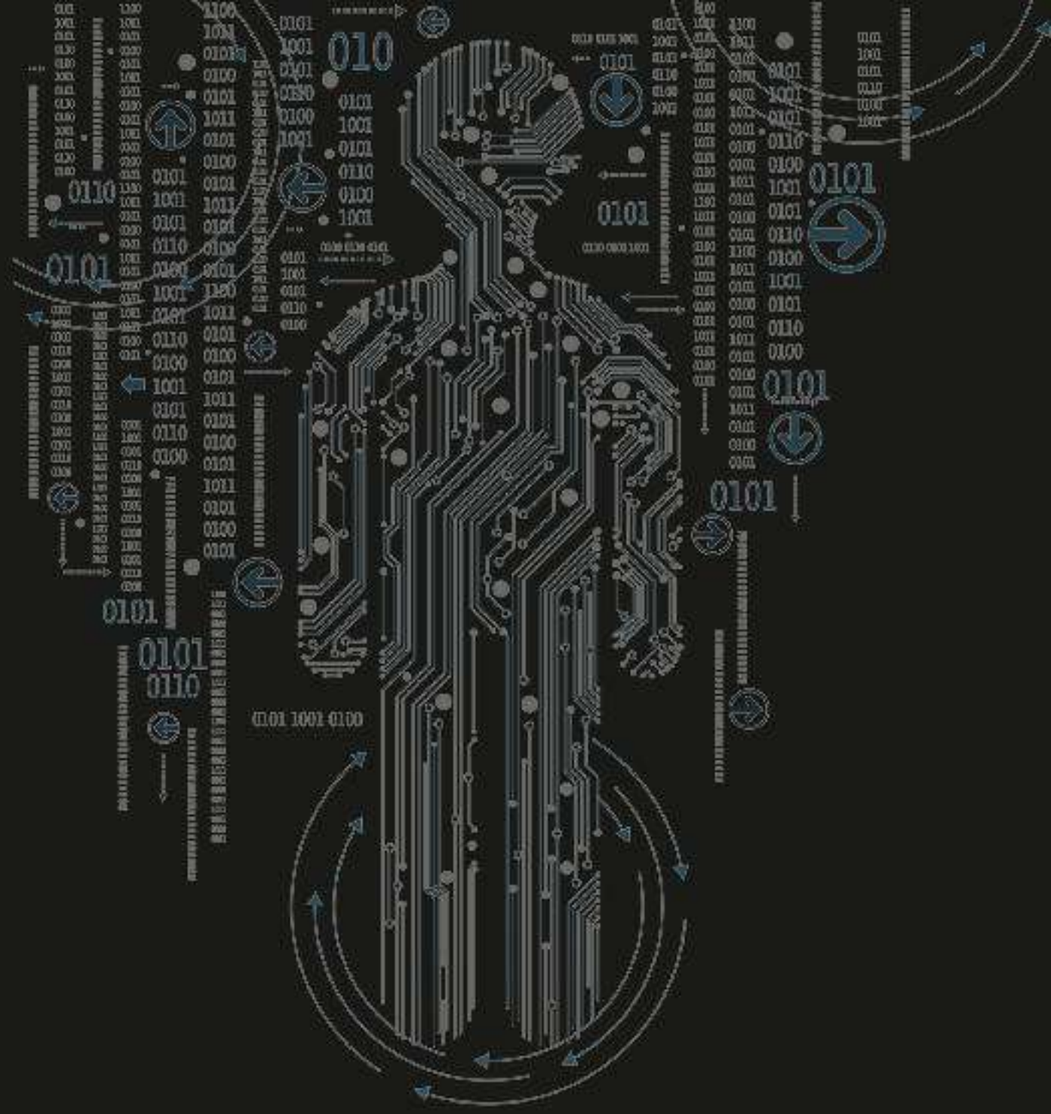
D. 22

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Structures & Unions



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Pointers & arrays

- Pointer to an element of array : *p
- Pointer to whole array (*p)[ ]
- Array of Pointers *p[ ]

BYJU'S EXAM PREP

** ppz ⇒ *(value at address A_{10})
= *(A_1)
= value at address (A_1) = 4

Q4. What is printed by the following C program?

```
#include <stdio.h>

int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
  z = **ppz;
  *py += 2;
  y = *py;
  x += 3;
  return x + y + z;
}

void main()
{
  int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  printf( "%d", f(c,b,a));
  getchar();
}
```

x: ~~4~~ 7 | py: A_1 | ppz: A_{10}

y: 7 | z: 5

7+7+5 = 19

c: 4 (A_1, ~~4~~ 7) | b: A_1 (A_{10}) | a: A_{10}

19

[GATE 2008]

A. 18

B. 19

C. 21

D. 22

BYJU'S EXAM PREP

Structure

struct c_class_info
{
Integer → int No.of Students Present;
string → char Concepts covered[i]
float → float Percentage of syllabus;
char Students_Names[ ],
}
Class1, class2, class3 --- class30

A structure / union is defined as group of dissimilar data items.

"struct" keyword for structure and "union" keyword for union is used to declare.

Syntax:

struct/union name

{

Data1;

Data 2;

.....

}; Variables

BYJU'S EXAM PREP

→ Derived datatypes

int x;
x is of type integer

Struct ABC x;
x is of type structure ABC.

Members and Variables

The Data defined inside structure or union is called as members of structure / union.

Structure or union is a data type (Derived) also. So, Variables of type structure / union can be created either along with declaration or later.

Syntax:

struct/union name{
Data member 1;
Data member 2;
...
}variables;

Members

Same

OR

struct /union variables;

union/Struct xyz
{ int a;
char b;
float c;
} v1, v2;

[OR]

union/Struct xyz
{ int a;
char b;
float c;
};
⋮
void main()
{
union/Struct xyz v1, v2;

BYJU'S EXAM PREP

Members and Variables

• (Dot) & → (Arrow) operators
— Member access operators.

Like for other data types, structure variables also can be normal variables or pointer variables.

To access members of structure / union,

Dot (.) operator is used for normal variable members. **variable.member;** $v_1.a / v_1.b / v_1.c$

Arrow(->) operator is used for pointer variable members. **variable->member;** $v_2 \to a;$ $v_2 \to b;$ $v_2 \to c;$

```
struct xyz
{ int a;
  char b;
  float c;
} v1;
```

```
struct xyz *v2;
```

v_1

a	b	c

v_2

a	b	c

scanf("%d", a) → Error
Printf("%c %d", b, c); → Error

Nested Structures / unions

It is valid to define a structure / union inside another structure /union, is said to be Nested structures / Unions.

```
struct{
...
union{
....
}
}
```

```
struct{
.....
struct{
....
}
}
```

```
union{
.....
struct{
.....
}
}
```

```
union{
.....
union{
.....
}
}
```

Let 2 Bytes per integer

```
struct XYZ
{   int a;   -> 2 Bytes
    char b;  -> 1 Byte
    float c; -> 4 Bytes
} v1, v2;
```

$\Sigma = 7$ Bytes

For Each variable of a structure

Memory = Σ members space.

v_1 (7 Bytes)

2B	1B	4B
a	b	c

v_2 (7 Bytes)

2B	1B	4B
a	b	c

```
union XYZ
{   int a;   -> 2 Bytes
    char b;  -> 1 Byte
    float c; -> 4 Bytes
} v1, v2;
```

Max = 4 Bytes

For Each union variable = Max { members space }

v_1 (4 Bytes)

v_2 (4 Bytes)

```
scanf("%d", &v1.a);   <- 15
printf("%d", v1.a);   <- 15
```

```
scanf("%d", &v1.a);
scanf("%c", &v1.
```

```
scanf("%d", &v1.a);   ← 15
scanf("%c", &v1.b);   ← #
scanf("%f", &v1.c);   ← 1.491
Printf("%d", v1.a);   ← Garbage (wrong)
Printf("%c", v1.b);   ← Garbage ( " )
Printf("%f", v1.c);   ← 1.491 (correct)
```

v1 (4 Bytes)

~~1.15~~ 491

Recent inputs/modifications only preserve-

Let- 2 Bytes per integer

```
Struct ABC
{ int a;   <- Member1
  Struct XYZ
  { float b;   = 4Bytes
    int c;     = 2Bytes
    char d[10];  = 10B
  } v1, v2;   16B
} v3, v4, v5;
```

Member2

16+16 = 32 Bytes

```
union ABC
{ int a;   = 2
  union XYZ
  { float b;   = 4
    int c;     = 2
    char d[10];  = 10
  } v1, v2;
} v3, v4, v5;
```

20

Max{20, 2} = 20

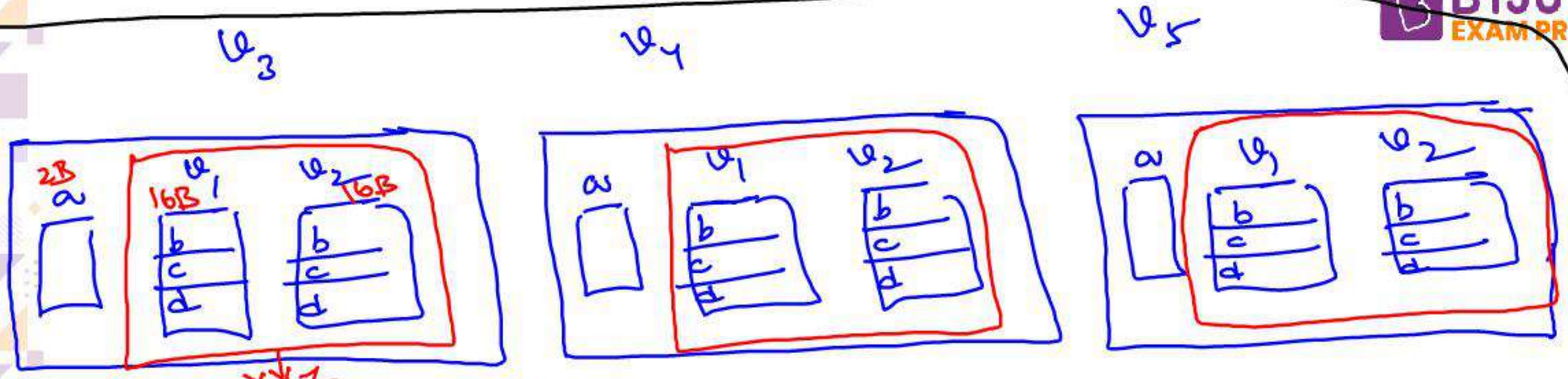
```
Struct ABC
{ int a;   = 2
  union XYZ
  { float b;
    int c;
    char d[10];
  } v1, v2;
} v3, v4, v5;
```

20

```
union ABC
{ int a;   = 2B
  Struct XYZ
  { int c;
    float b;
    char d[10];
  } v1, v2;
} v3, v4, v5;
```

32

Size of (v_1) == Size of (v_2) = 16	10 Bytes	10 Bytes	16 Bytes
Size of (v_3) = (v_4) = (v_5) = 34 Bytes	20 Bytes	22 Bytes	32 Bytes
Size of (XYZ) = 32 Bytes	10+10 = 20 Bytes	20 Bytes	16+16 = 32B
Size of (ABC) = 102 Bytes	20+20+20 = 60 Bytes	66 Bytes	32+32+32 = 96B



XYZ

(16+16 = 32 Bytes)

$v_3 = 2 + 32 = 34B$

$v_4 = 2 + 32 = 34$

$v_5 = 2 + 32 = 34$

NOTE:

Structure | union size = Σ variables size

Variable size { Structure — Σ members
union — Max (members)

ABC.

Size of (ABC) = 34 + 34 + 34 = 102 Bytes.

BYJU'S EXAM PREP

Self – Referential Structure

To be contd--- in Next class

A structure whose member is variable of itself is called as self referential structure.

Syntax:

```
struct Structure{

.....

struct Structure *variable;

}
```

BYJU'S EXAM PREP

Structure Vs Union

please read.

Structure	Union
When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a structure at a time.	In union, only the first data member can be initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.

BYJU'S EXAM PREP

Example - 1

```
#include <stdio.h>
struct S {
int S;
};
int main(void) {
struct S S;
S.S = sizeof(struct S) / sizeof(S);
printf("%d",S.S);
return 0;
}
```

BYJU'S EXAM PREP

Example - 2

```
#include <stdio.h>
#include <string.h>
struct Q {
char S[3];
};
struct S {
struct Q Q;
};
int main(void) {
struct S S = { '\0', '\0','\0' };
S.Q.S[0] = 'A';
S.Q.S[1] = 'B';
printf("%d",strlen(S.Q.S));
return 0;
}
```

BYJU'S EXAM PREP

Example - 3

```
#include <stdio.h>
struct Q {
int a,b,c;
};
struct S {
int a,b,c;
struct Q Q;
};
int main(void) {
struct Q Q = { 6,5,4 };
struct S S = { 4,5,6 };
S.Q = Q;
printf("%d",S.b - S.Q.b);
return 0;
}
```

BYJU'S EXAM PREP

Example - 4

```
#include <stdio.h>
#include <stdlib.h>
struct S {
int a;
struct S *b;
};
int main(void) {
struct S *x = (struct S*) malloc(sizeof(struct S));
struct S *y = (struct S*) malloc(sizeof(struct S));
x->a = 2;
x->b = y;
y->a = 4;
y->b = x;
printf("%d",x->b->b->b->b->a);
free(x); free(y);
return 0;
}
```

BYJU'S EXAM PREP

Example - 5

```
#include<stdio.h>
int main()
{
struct num
{
int i, j, k, l;
};
struct num n = {1, 2, 3};
printf("%d %d %d %d", n.i, n.j, n.k, n.l);
}
```

A. 1 2 3 garbage value

B. 1 2 3 0

C. 1 2 3 3

D. None of the above

BYJU'S EXAM PREP

GATE 2000

Consider the following C declaration

```
struct (
short s[5];
union {
float y;
long z;
}u;
}t;
```

Assume that the objects of the type short, float and long occupy 2 bytes, 4 bytes and 8 bytes, respectively. The memory requirement for variable t, ignoring alignment consideration, is

(A) 22 bytes
(B) 18 bytes
(C) 14 bytes
(D) 10 bytes

BYJU'S EXAM PREP

GATE 2000

The following C declarations

```
struct node
{
   int i;
   float j;
};
struct node *s[10] ;
```

define s to be

(A) An array, each element of which is a pointer to a structure of type node

(B) A structure of 2 fields, each field being a pointer to an array of 10 elements

(C) A structure of 3 fields: an integer, a float, and an array of 10 elements

(D) An array, each element of which is a structure of type node.

BYJU'S EXAM PREP

GATE 2018

Consider the following C program.

```
#include <stdio.h>
struct Ournode {
  char x, y, z;
};
int main() {
  struct Ournode p = {'1', '0', 'a' + 2};
  struct Ournode *q = &p;
  printf("%c, %c", *((char *)q + 1), *((char *)q + 2));
  return 0;
}
```

The output of this program is:

(A) 0, c

(B) 0, a+2

(C) '0', 'a+2'

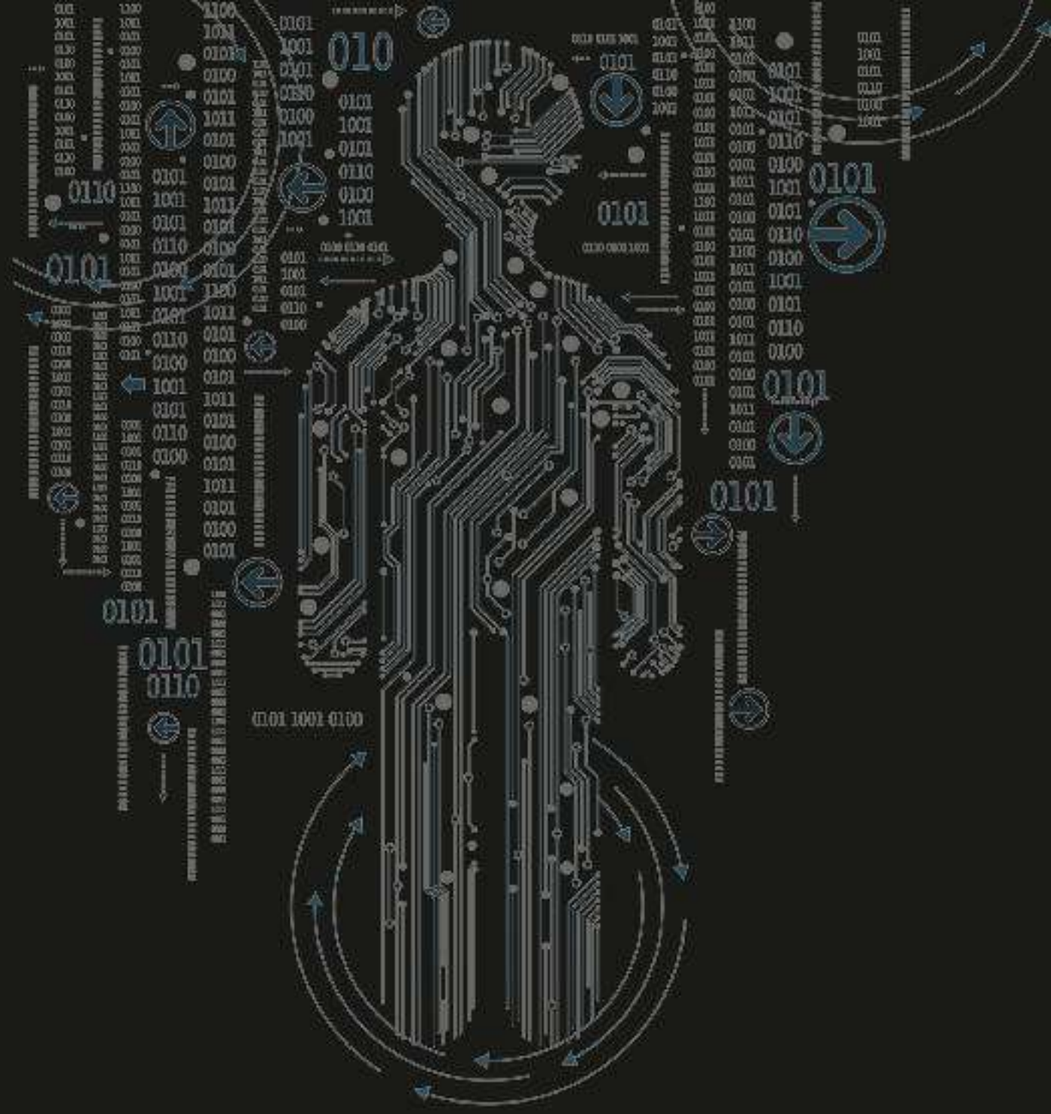
(D) '0', 'c'

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Linked Lists - 1



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

define var 3;

Short num[3][2] = { 3, 6, 9, 12, 15, 18 }

	0	1
num+0	2000 3	2003 6
→ num+1	2006 9	2009 12
num+2	2012 15	2015 18

3
6
9

x[y] ⇒ x+y

(num+1)[1]

(num+1)+1
= num+2

12
15
18

15

*(num+1)[1] ⇒ *(num+1)[1]

**(num+2) ⇒ *(*(num+2)+0)

*(num+2)
= value at address 2012
= num[2][0]
= 15

O/P: 12, 15

BYJU'S EXAM PREP

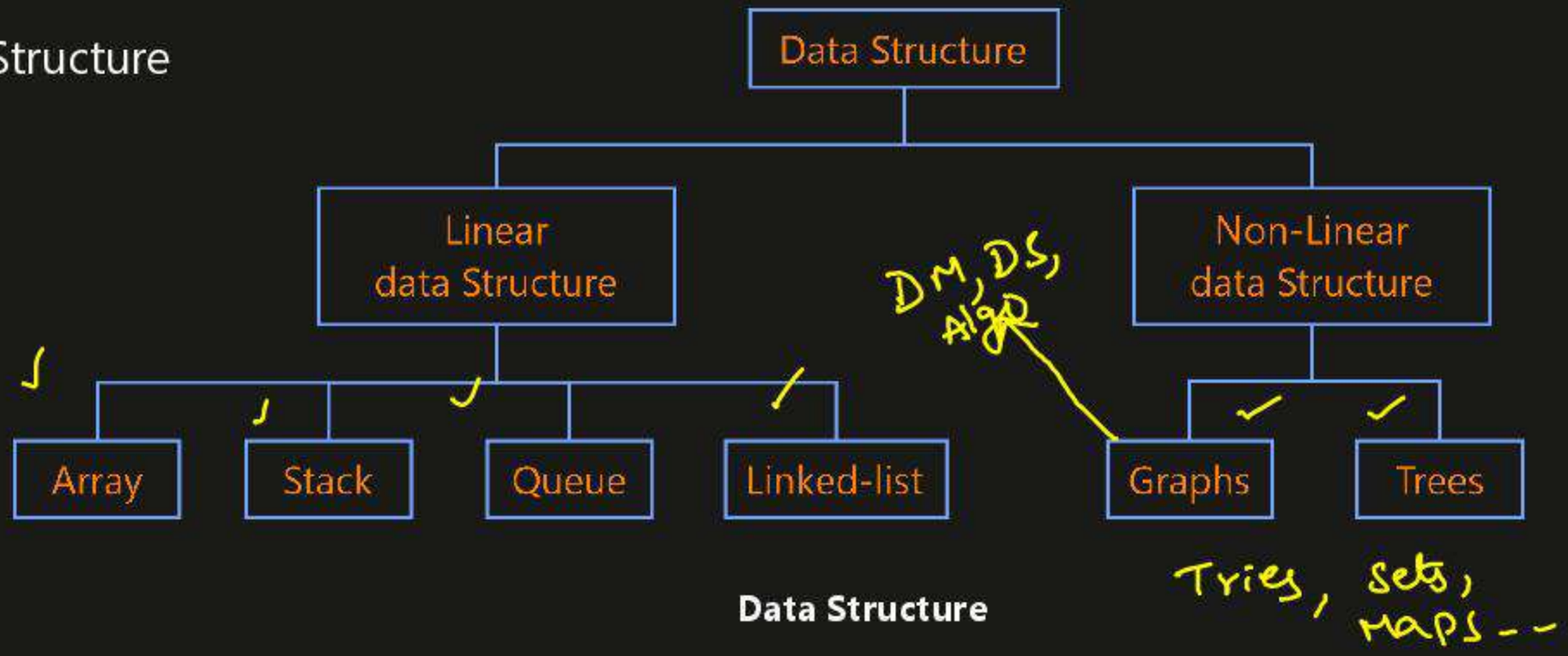
Read write search create ---

Data Structure

How operations are carried out on data?

Data Structure defined the way of managing data in memory. There are 2 forms of data structures:

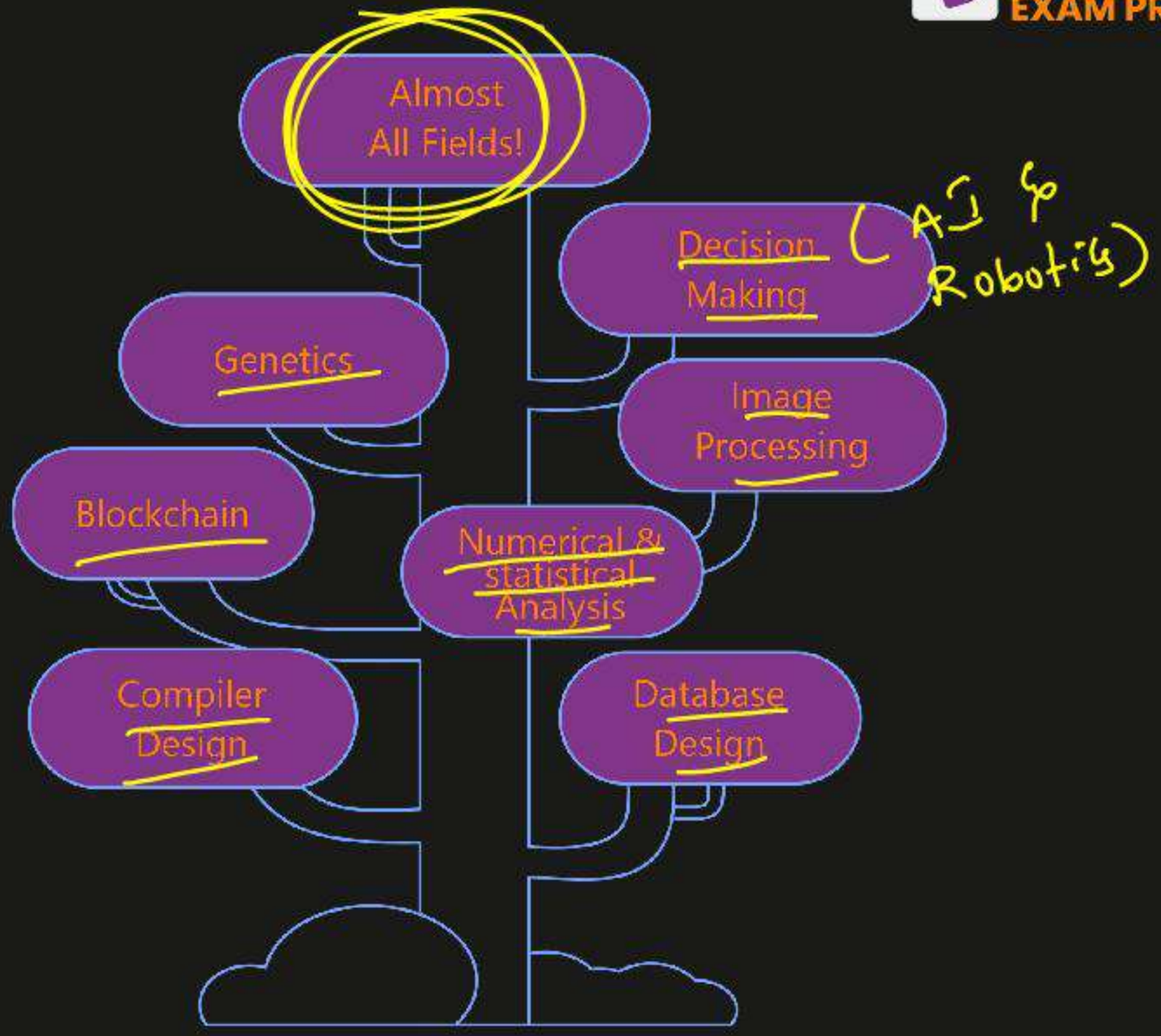
- Linear Data Structure
- Non – Linear Data Structure



Data Structure

BYJU'S EXAM PREP

Applications



Application of Data Structures

BYJU'S EXAM PREP

Linear Data Structure

If the elements of a data structure result in a sequence or a linear list then it is called a linear data structure.

Example: Arrays, Linked List, Stacks, Queues.

Linked Lists - 1

Individual | Independent

Linked List

Linked list is a linear data structure that includes a series of connected nodes. Linked list can be defined as the nodes that are randomly stored in the memory. A node in the linked list contains two parts, i.e., first is the data part and second is the address part. (Connection Part)

There are 3 types of linked lists:

- Singly Linked List (SLL)
- Doubly Linked List (DLL)
- Circular Linked List (CLL)

BYJU'S EXAM PREP

Operations On Linked List

The basic operations that are supported by a linked list are mentioned as follows -

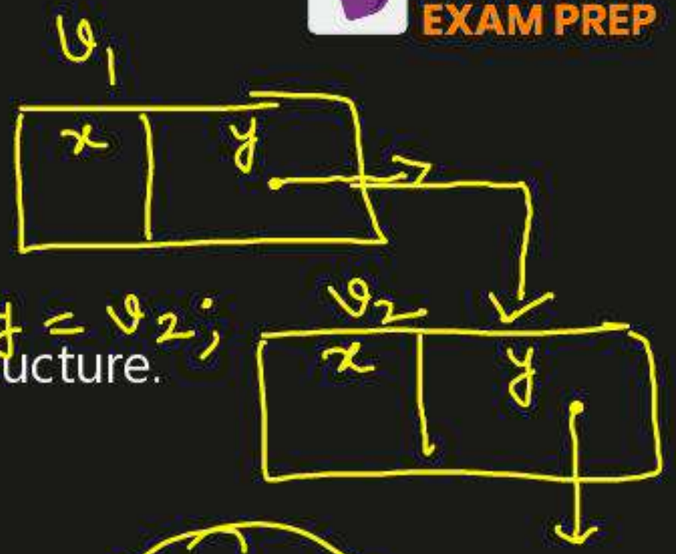
- Insertion - This operation is performed to add an element into the list. (Append)
- Deletion - It is performed to delete an operation from the list. (Modifying / updation / write)
- Display - It is performed to display the elements of the list. (Traverse / Access / Read)
- Search - It is performed to search an element from the list using the given key. (Find)

BYJU'S EXAM PREP

Self – Referential Structure

```
struct ABC
{   int x;
    struct ABC *y;
} *v1, *v2;
```

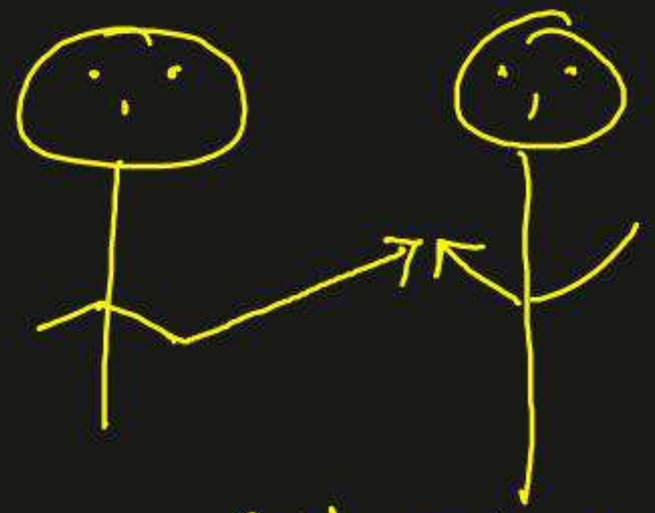
$v_1 \rightarrow y = v_2;$



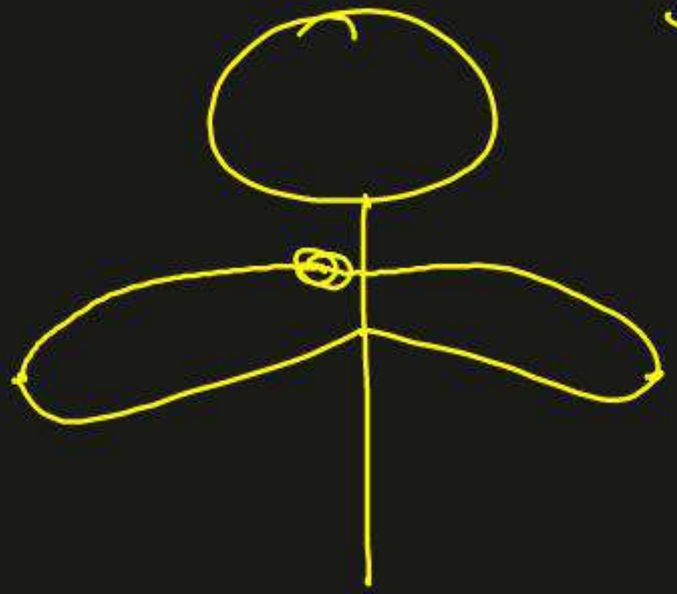
A structure whose member is variable of itself is called as self referential structure.

Syntax:

```
struct Structure{

.....

struct Structure *variable;

}
```



Self reference.



Recursion

Linked Lists - 1

Singly Linked List (SLL)

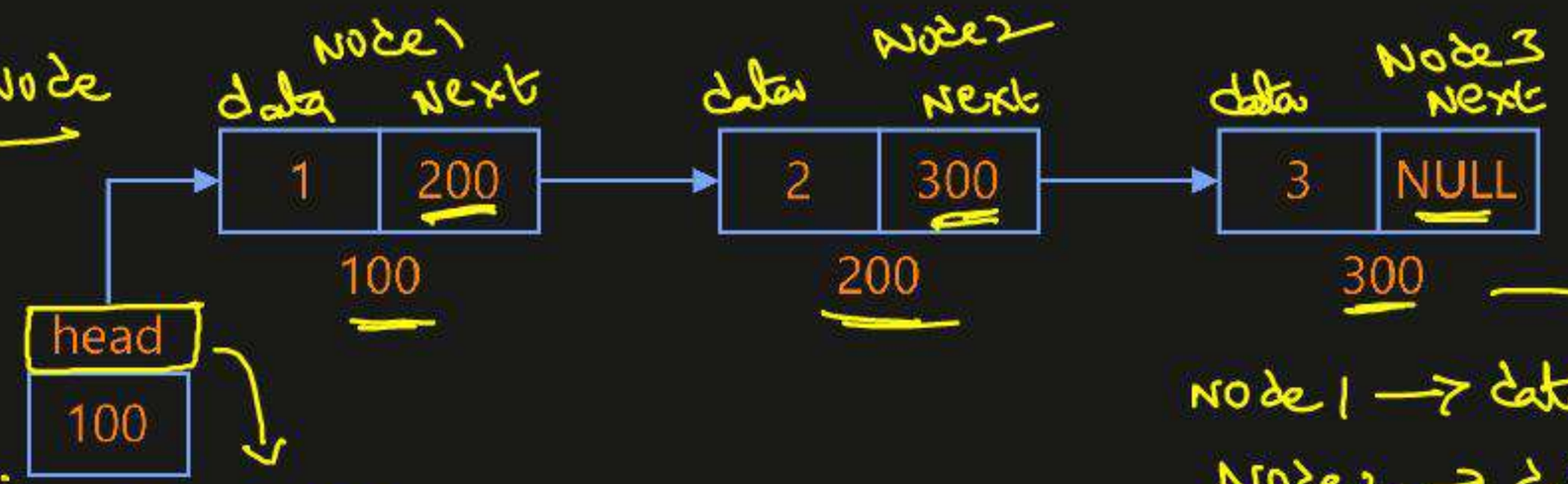
Struct ABC
{ int x;
} Struct ABC *y;

int *p;
Struct SLL *head;

In SLL, each node points to the next node of the list. The representation of the linked list is shown below :

Creation of Node

Struct SLL
{ int data;
Struct SLL *next;
} *node1, *node2,
*node3;
Struct SLL *head;



Beginning of list.

Singly Linked List (SLL)

Address

node1 → data = 1;
node2 → data = 2;
node3 → data = 3;

node1 → next = node2;
node2 → next = node3;
node3 → next = NULL;

head = node1;

BYJU'S EXAM PREP

Insertion into SLL

The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

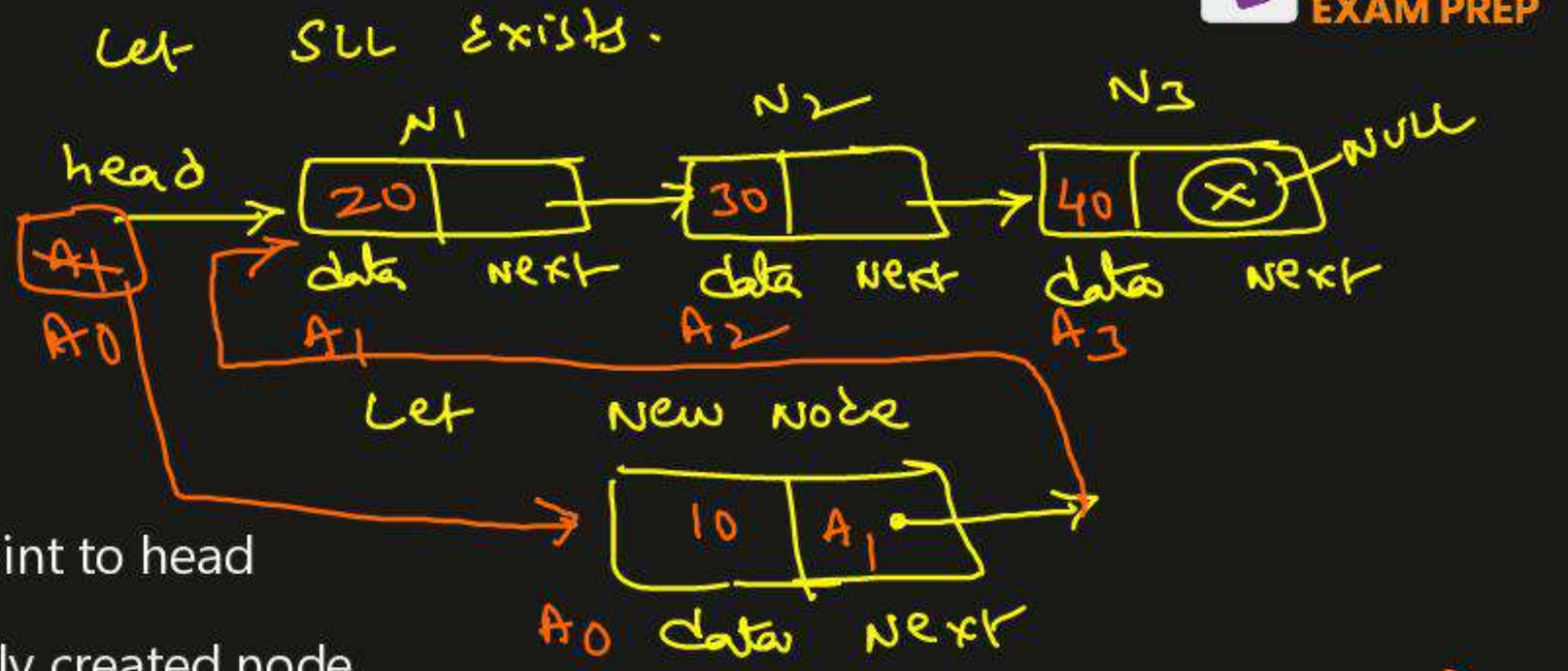
1. Insertion at beginning
2. Insertion at end of the list
3. Insertion after specified node

BYJU'S EXAM PREP

Insertion at beginning

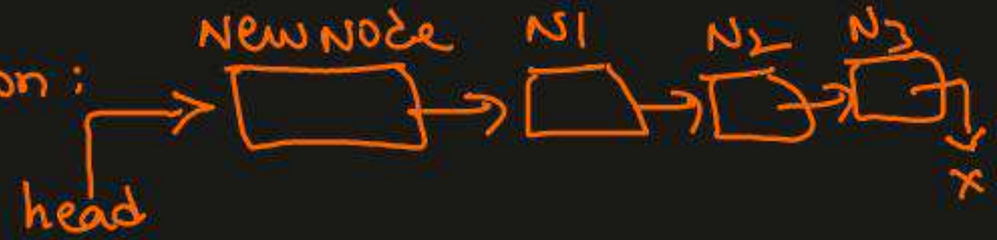
1. Allocate memory for new node
2. Store data
3. Change next of new node to point to head
4. Change head to point to recently created node

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;
newNode->next = head;
head = newNode;
```



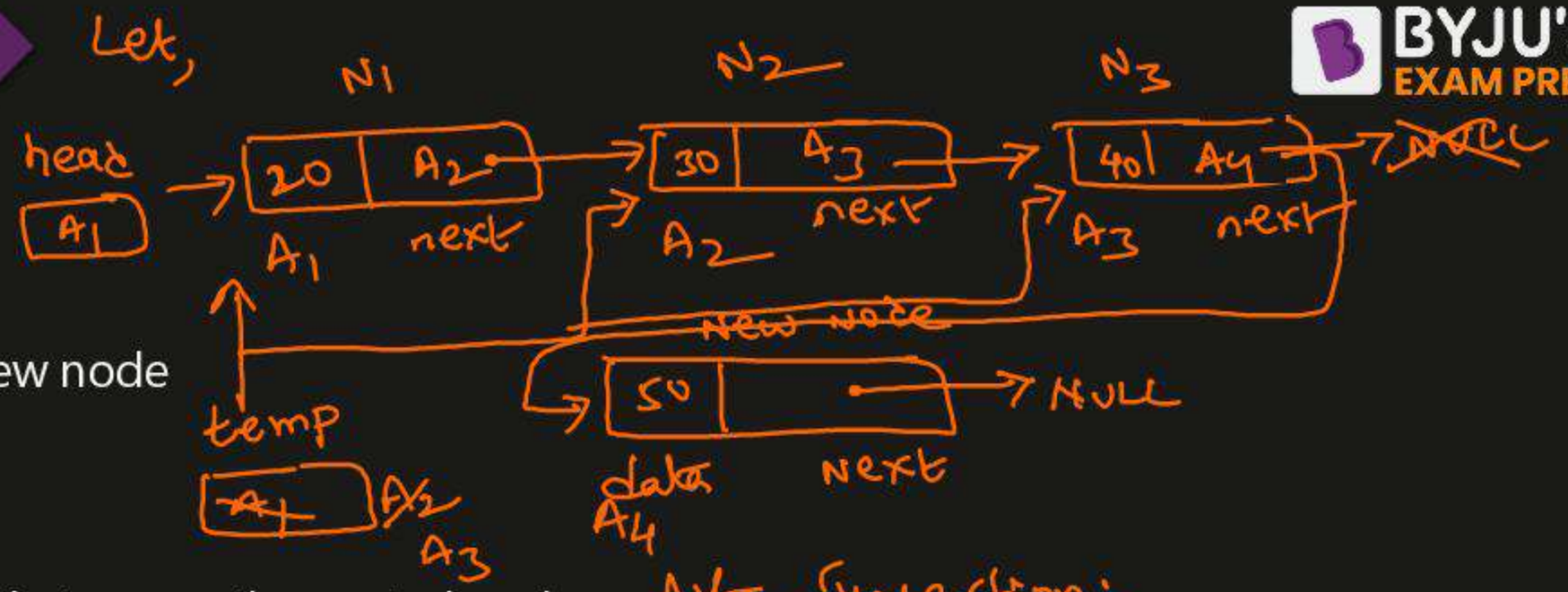
↳ 10 scanf("%d", &newNode->data);

After Insertion:



BYJU'S EXAM PREP

Insertion at Last



1. Allocate memory for new node
2. Store data
3. Traverse to last node
4. Change next of last node to recently created node

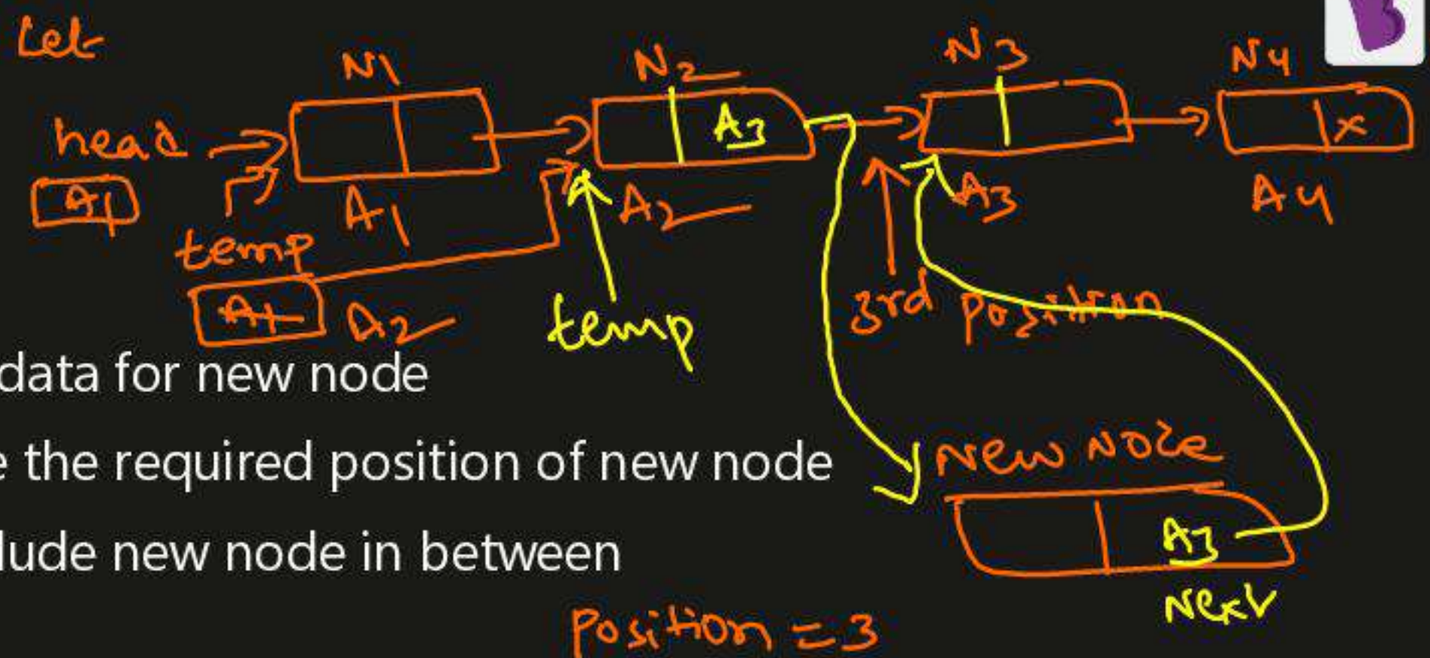
After Insertion:

head → N1 → N2 → N3 → New node → NULL

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;(OR) Scanf("--");
newNode->next = NULL;
struct node *temp = head;
while(temp->next != NULL){
  temp = temp->next;
}
temp->next = newNode;
```

BYJU'S EXAM PREP

Insertion at Middle

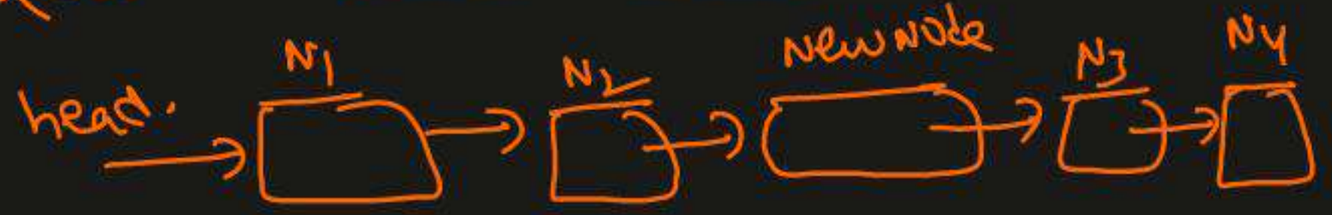


1. Allocate memory and store data for new node
2. Traverse to node just before the required position of new node
3. Change next pointers to include new node in between

```
struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = value;
struct node *temp = head;
for(int i=2; i < position; i++) {
  if(temp->next != NULL) {
    temp = temp->next;
  }
}
newNode->next = temp->next;
temp->next = newNode;
```

i=2 2<3
i=3
3<3
False

After Insertion



(i=1; i<(Position-1); i++)

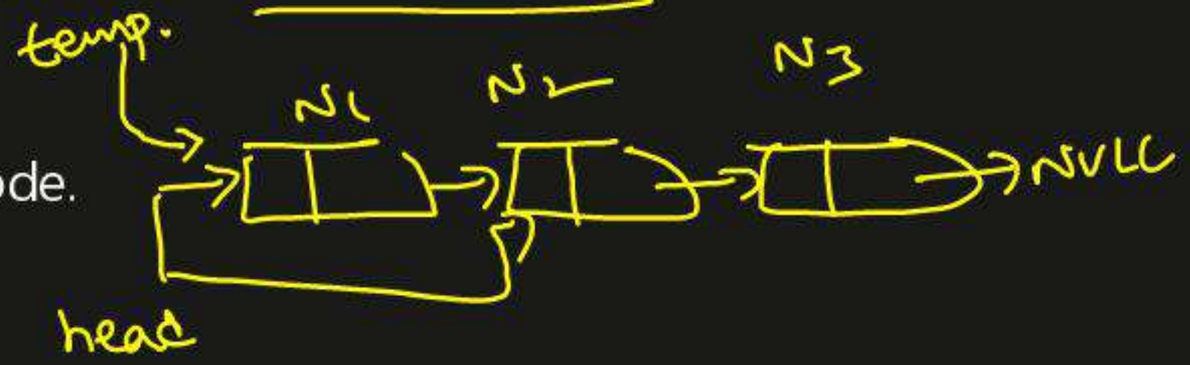
BYJU'S EXAM PREP

Deletion

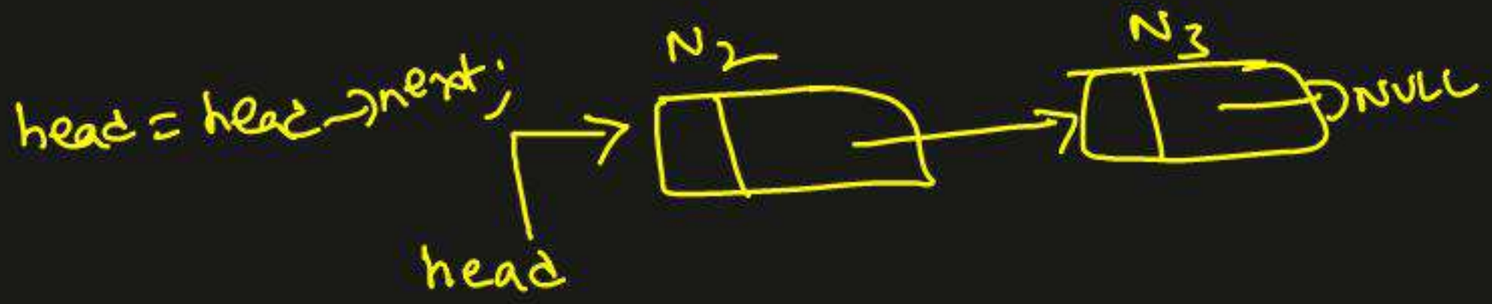
You can delete either from the beginning, end or from a particular position.

1. Delete from beginning
   - Create a temp pointer, point to first node.
   - Point head to the second node
   - Free temp.

```
temp=head;
head = head->next;
free(temp);
```

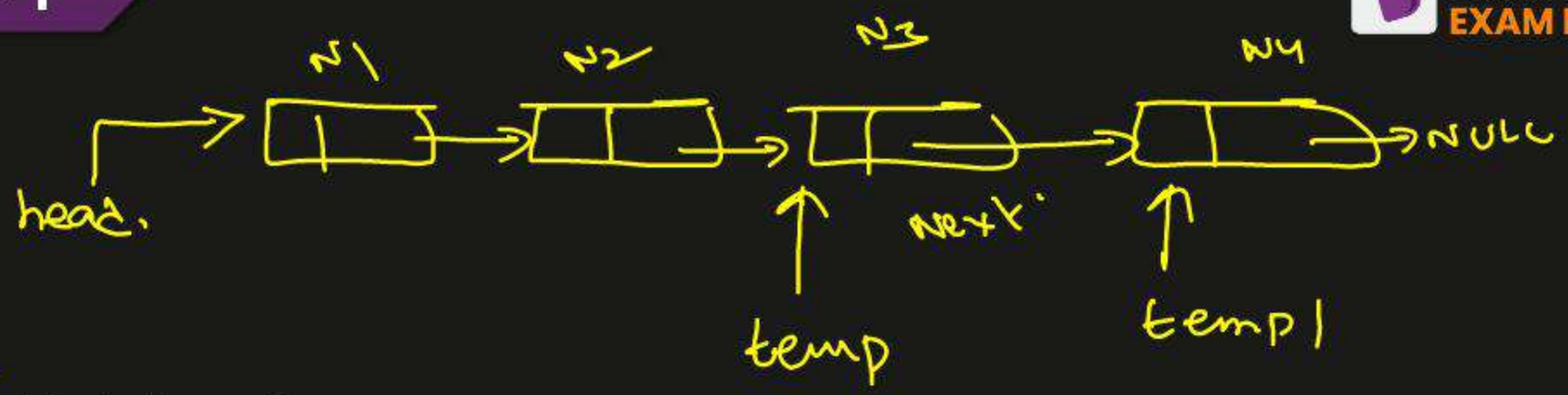


After Deletion :



BYJU'S EXAM PREP

Deletion



2. Delete from end

Traverse to second last element

Change its next pointer to null

```
struct node* temp = head;, *temp1;
while(temp->next->next!=NULL){
  temp = temp->next;
}    temp1 = temp->next;
temp->next = NULL;
:    free(temp1);
```

BYJU'S EXAM PREP

Deletion

3. Delete from middle

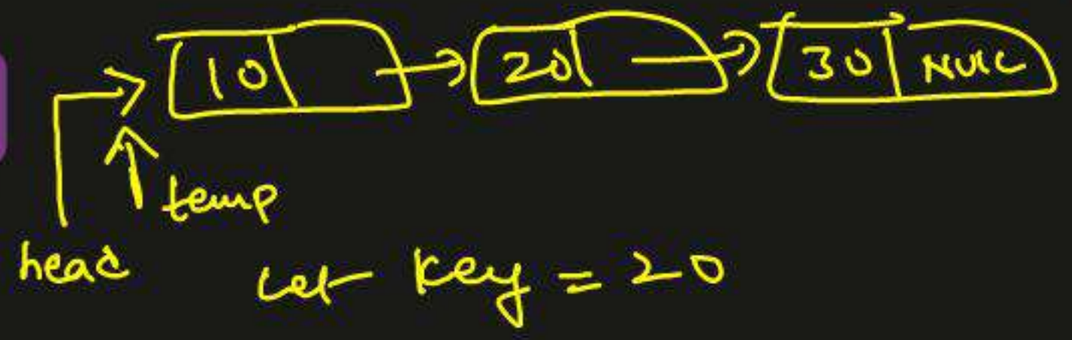
Traverse to element before the element to be deleted

Change next pointers to exclude the node from the chain

```
for(int i=2; i< position; i++) {
  if(temp->next!=NULL) {
    temp = temp->next;
  } }  temp1 = temp -> next;
temp->next = temp->next->next;
          free (temp1);
```

BYJU'S EXAM PREP

Search an Element on a Linked List



- We are finding item on a linked list.
- Make head as the current node.
- Run a loop until the current node is NULL because the last element points to NULL.
- In each iteration, check if the key of the node is equal to item. If it the key matches the item, return true otherwise return false.

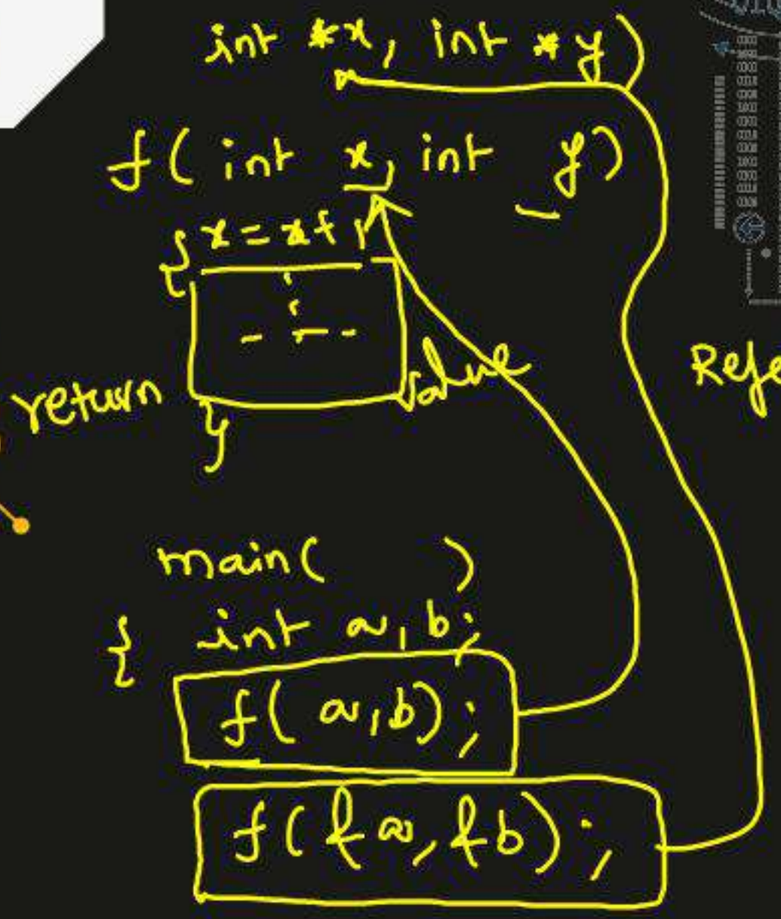
```
while (temp->next != NULL)
{ if (temp->data == key)
      search success
  else
      failed;
}
```

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Linked Lists - 2



```
int f(int x, int y)
{   int z;
    z=x+1;
    return z;
}
void main( )
{   int a=5, b=7;
    Printf("%d", f(a,b));
}
```

return value : 6

```
int f(int *x, int *y)
{   int *z;
    z= x+1;
    return z;
}
void main( )
{   int a=5, b=7;
    Printf("%d", f(a,b
```

1004
1008
1012

loop-

```
int z;
z= *x+1;
return z;
```

6 7 8 9 --

return value 1004 / 6

ABOUT ME

- Teaching Experience: 14+ Years
- Achievements: B.Tech, M.Tech, (Ph.D.)
- Achievements:
  - CS Senior Faculty
  - Mentored More Than 30,000+ Students
  - GATE Qualified
  - UGC-NET Qualified
  - SET Qualified
  - CISCO Certified

SATYA NARAYANA KAITEPALLI

Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

- L.L is a linear DS, Nodes are connected.
- Node — data + address
- 3 types of linked lists : SLL, DLL, CLL
- Operations : Creation of node, Insert, Delete, traverse, Search
- Creation of SLL Node:

```
Struct SLL
{ data;
  Struct SLL *next;
};
```

Insertion / Deletion
- At Beginning
- At End
- At middle Node

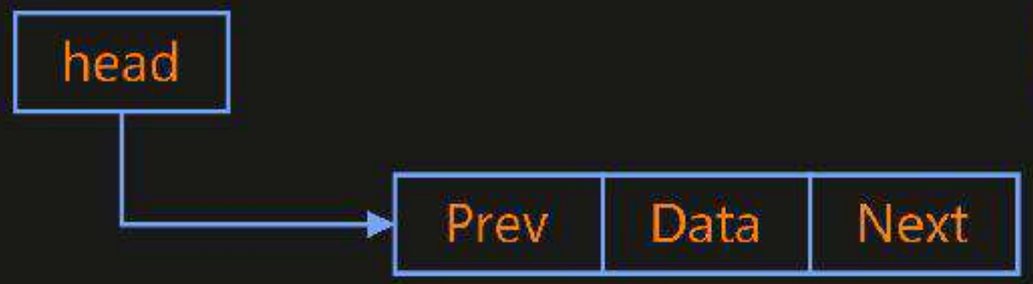
BYJU'S EXAM PREP

Doubly Linked List

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.

Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).

A sample node in a doubly linked list is shown in the figure.



Node

```
struct DLL
{ struct DLL *prev;
  data;
  struct DLL *next;
};
```

BYJU'S EXAM PREP

Doubly Linked List Node

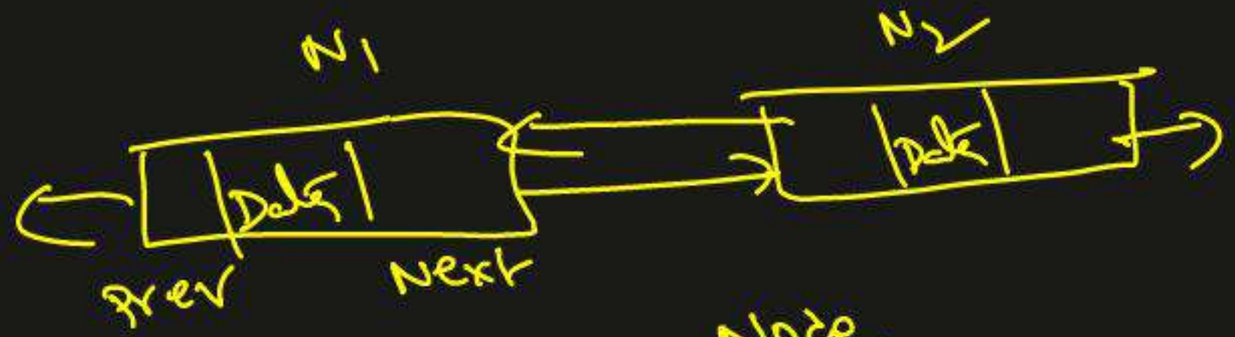
In C, structure of a node in doubly linked list can be given as :

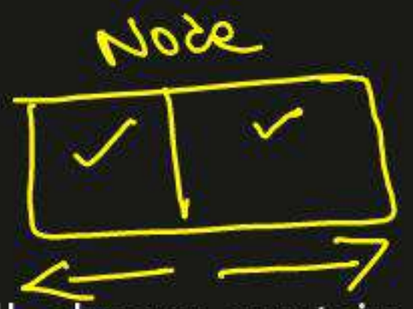
$N_1 \rightarrow next = N_2$

$N_2 \rightarrow prev = N_1$

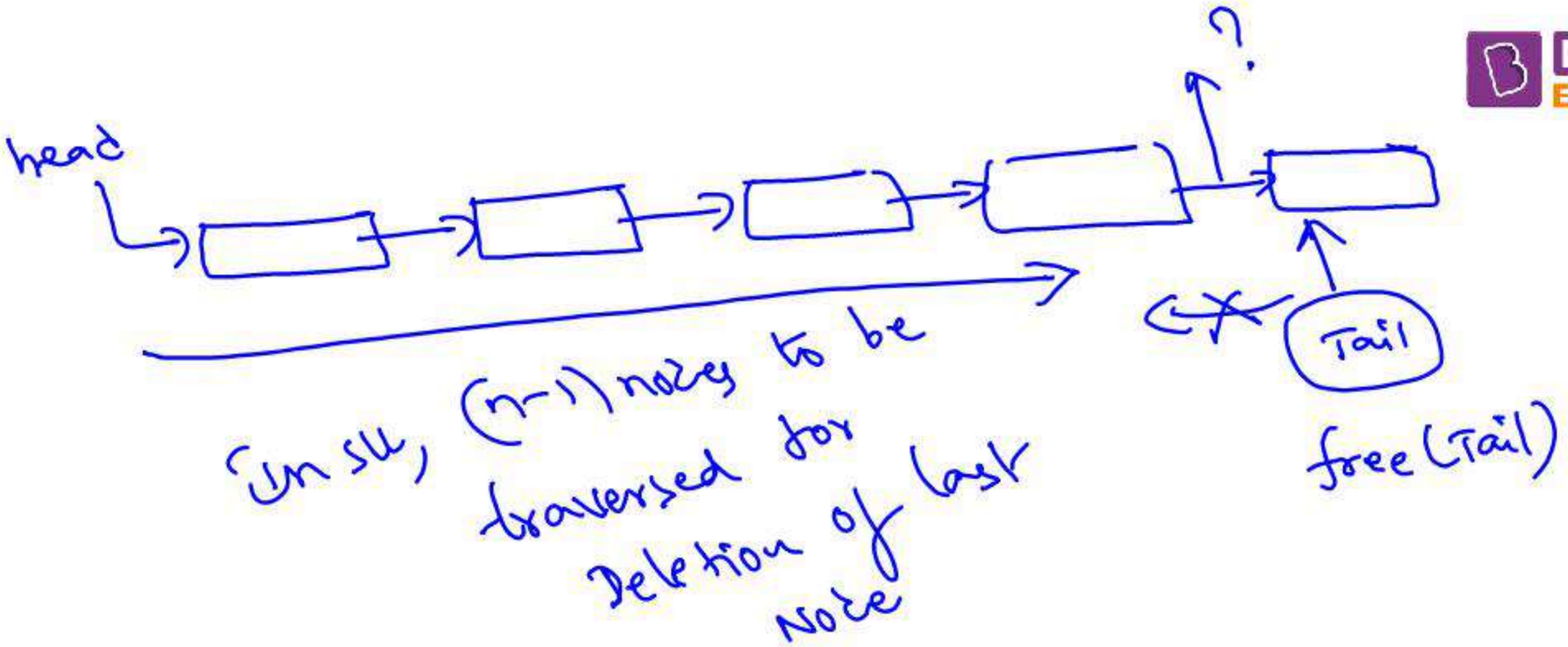
```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
```

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





The prev part of the first node and the next part of the last node will always contain null indicating end in each direction.


head
?
Tail
free(Tail)
In SLL, (n-1) nodes to be traversed for Deletion of last Node

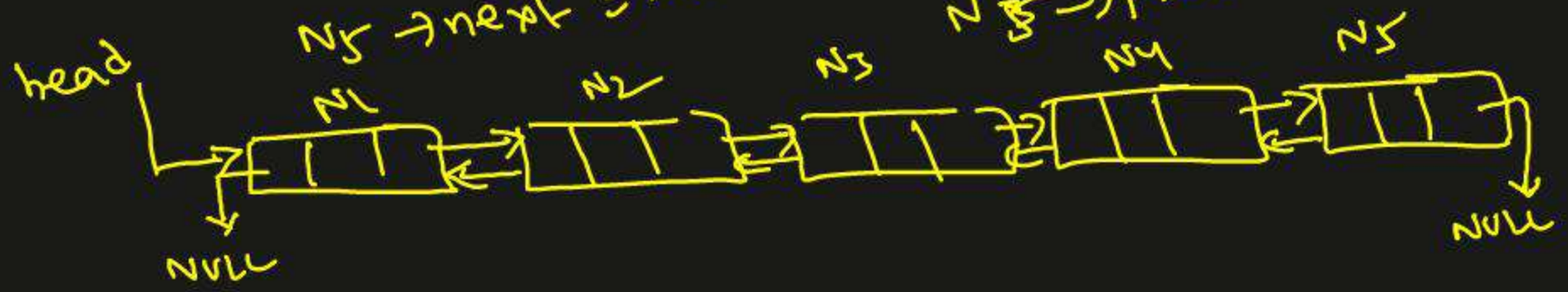
BYJU'S EXAM PREP

Operations

Node Creation :

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

```
struct node *head;
struct node *N1, *N2, *N3 --- N5;

head = N1;
N1 -> next = N2;
N2 -> next = N3;
:
N5 -> next = NULL

N1 -> prev = NULL
N2 -> prev = N1
N3 -> prev = N2
N4 -> prev = N3
N5 -> prev = N4.
```



BYJU'S EXAM PREP

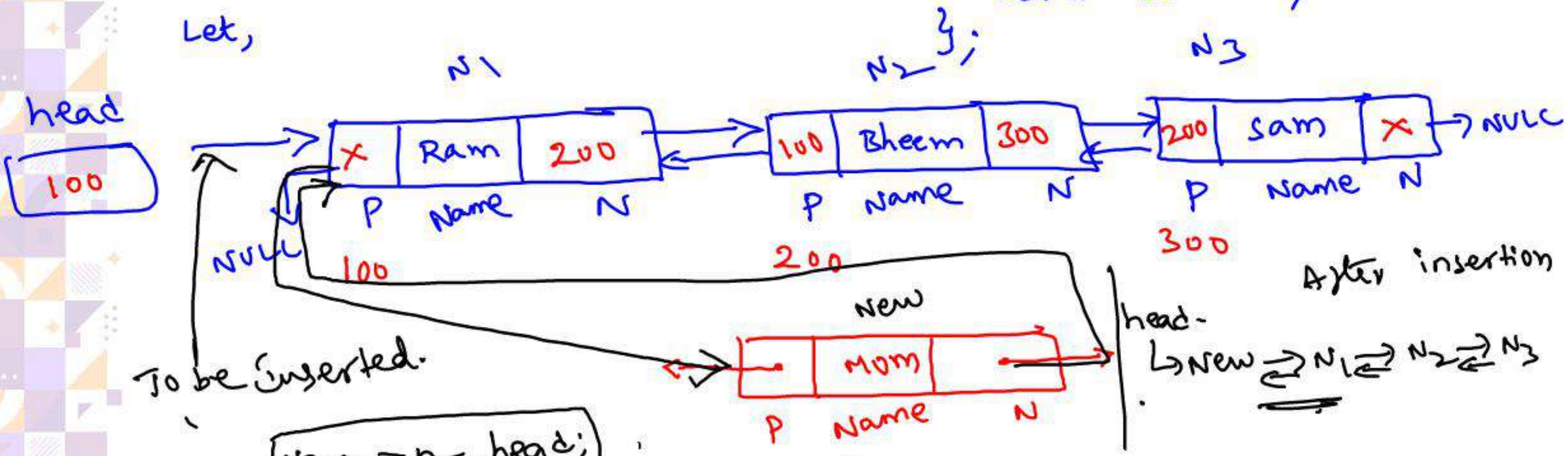
Operations

SNO	Operation	Description
1	Insertion at beginning	Adding the node into the linked list at beginning.
2	Insertion at end	Adding the node into the linked list to the end.
3	Insertion after specified node	Adding the node into the linked list after the specified node.
4	Deletion at beginning	Removing the node from beginning of the list
5	Deletion at the end	Removing the node from end of the list.
6	Deletion of the node having given data	Removing the node which is present just after the node containing the given data.
7	Searching	Comparing each node data with the item to be searched and return the location of the item in the list if the item found else return null.
8	Traversing	Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, display, etc.

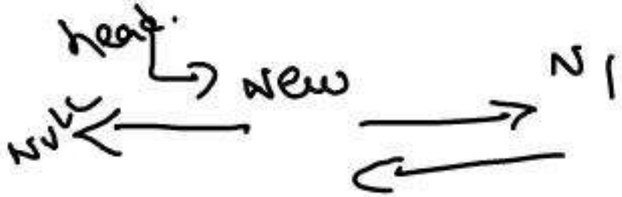
Insertion at Beginning

```
Struct DLL
{
    Struct DLL *P;
    Char Name[10];
    Struct DLL *N;
};
```

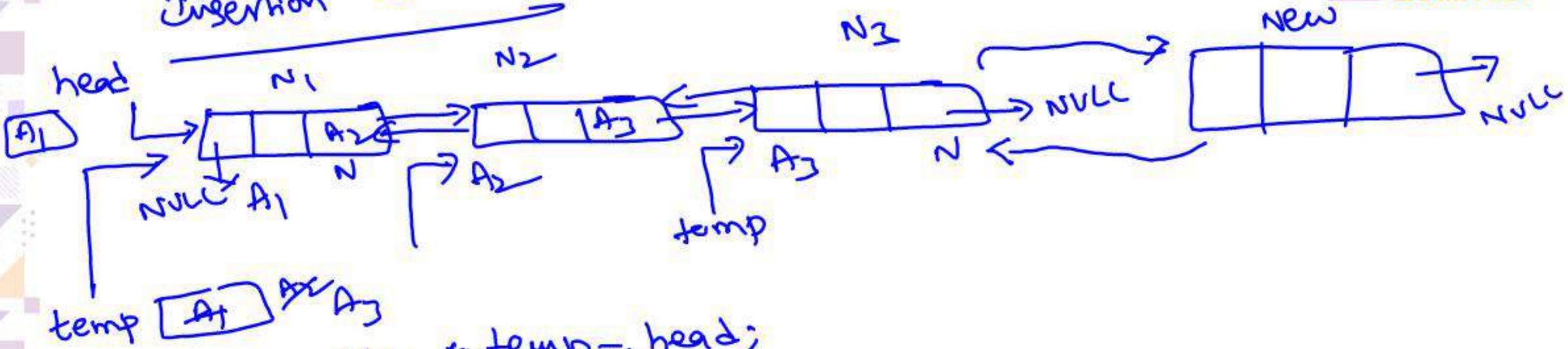
Let,




```
New -> n = head;
New -> n -> p = New;
head = new;
New -> p = NULL;
```



Insertion at last



temp [A1] ~~A2~~ A3

```
Struct DLL * temp = head;
    while ( temp->next != NULL )
        temp = temp -> Next;
```

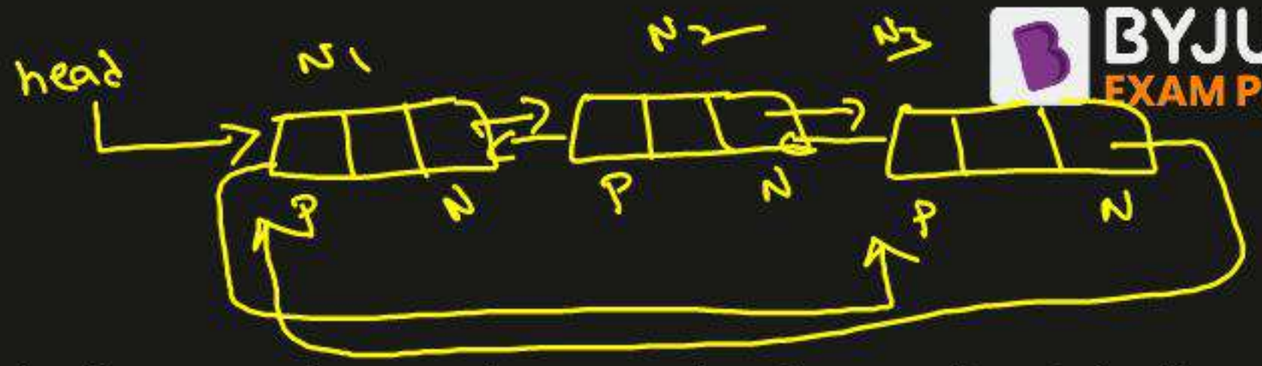
```
new -> n = NULL;
temp -> next = New;
New -> P = temp;
```

Linked Lists - 2

Circular Linked List

Singly

Doubly



In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.

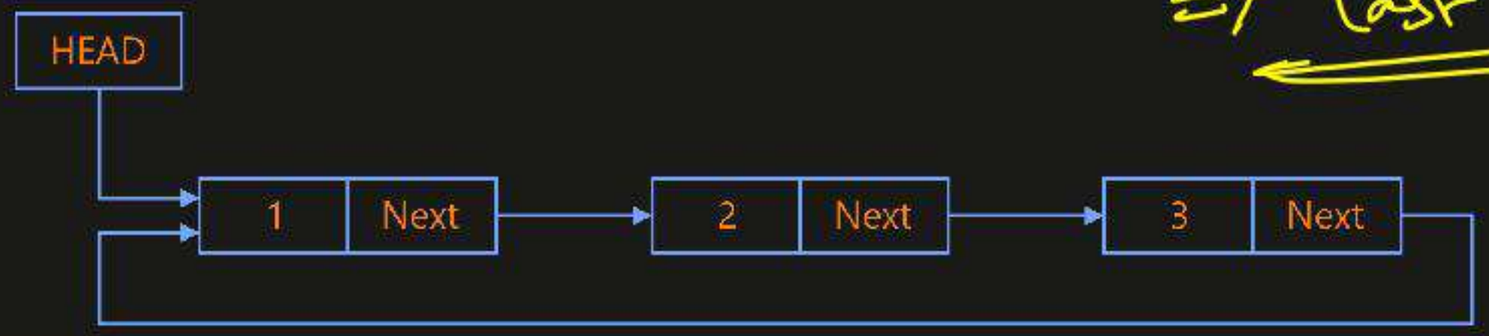
We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no ending. There is no null value present in the next part of any of the nodes.

if (temp→next == head)

⇒ last node

The following image shows a circular singly linked list.

CSLL



Circular Linked List

BYJU'S EXAM PREP

Circular Linked List

Circular linked list are mostly used in task maintenance in operating systems. There are many examples where circular linked list are being used in computer science including browser surfing where a record of pages visited in the past by the user, is maintained in the form of circular linked lists and can be accessed again on clicking the previous button.

BYJU'S EXAM PREP

Operations

- Creation Of Node
- Insertion
- Deletion
- Searching
- Traversing

Linked Lists - 2

To be Contd...

Q1. What does the following function do for a given Linked List with first node as head ?

```
void fun1(struct node* head)
{
    if(head == NULL)
    return;
    fun1(head->next);
    printf("%d  ", head->data);
}
```

A. Prints all nodes of linked lists

B. Prints all nodes of linked list in reverse order

C. Prints alternate nodes of Linked List

D. Prints alternate nodes in reverse order

Linked Lists - 2

Q2. What does the following function do for a given Linked List with first node as head?

```
void fun1(struct node* head)
{
    if(head->next == NULL)
    return;
    printf("%d  ", head->next->data);
    fun1(head->next);
}
```

A. Prints all nodes of linked lists

B. Prints all nodes of linked list except first node

C. Prints alternate nodes of Linked List

D. Prints all nodes of linked list except last node

BYJU'S EXAM PREP

Q3. In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is -

[GATE 2002]

A. log n

B. n/2

C. logn – 1

D. n

BYJU'S EXAM PREP

Q4. The following C function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution? [GATE 2008]

```
void rearrange(struct node *list)  {
  struct node *p, * q;
  int temp;
  if ((!list) || !list->next)        return;
  p = list;     q = list->next;
  while(q)     {
     temp = p->value;  p->value = q->value;
     q->value = temp;   p = q->next;
     q = p?p->next:0;     }  }
```

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

A. 1,2,3,4,5,6,7

B. 2,1,4,3,6,5,7

C. 1,3,2,5,4,7,6

D. 2,3,4,5,6,7,1

BYJU'S EXAM PREP

Q5. Let P be a singly linked list. Let Q be the pointer to an intermediate node x in the list. What is the worst-case time complexity of the best-known algorithm to delete the node x from the list ?

[GATE 2004 IT]

A. O(n)

B. $O(\log^2 n)$

C. O(logn)

D. O(1)

BYJU'S EXAM PREP

Q6. The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. [GATE 2010]

```
typedef struct node  {
  int value;
  struct node *next;
}Node;
Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
    return head;
  q = NULL; p = head;
  while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  _____________________________
  return head;
}
```

Linked Lists - 2

Choose the correct alternative to replace the blank line.

A. q = NULL; p->next = head; head = p;

B. q->next = NULL; head = p; p->next = head;

C. head = p; p->next = q; q->next = NULL;

D. q->next = NULL; p->next = head; head = p;

Linked Lists - 2

BYJU'S EXAM PREP

A. q = NULL; p->next = head; head = p;

B. q->next = NULL; head = p; p->next = head;

C. head = p; p->next = q; q->next = NULL;

D. q->next = NULL; p->next = head; head = p;

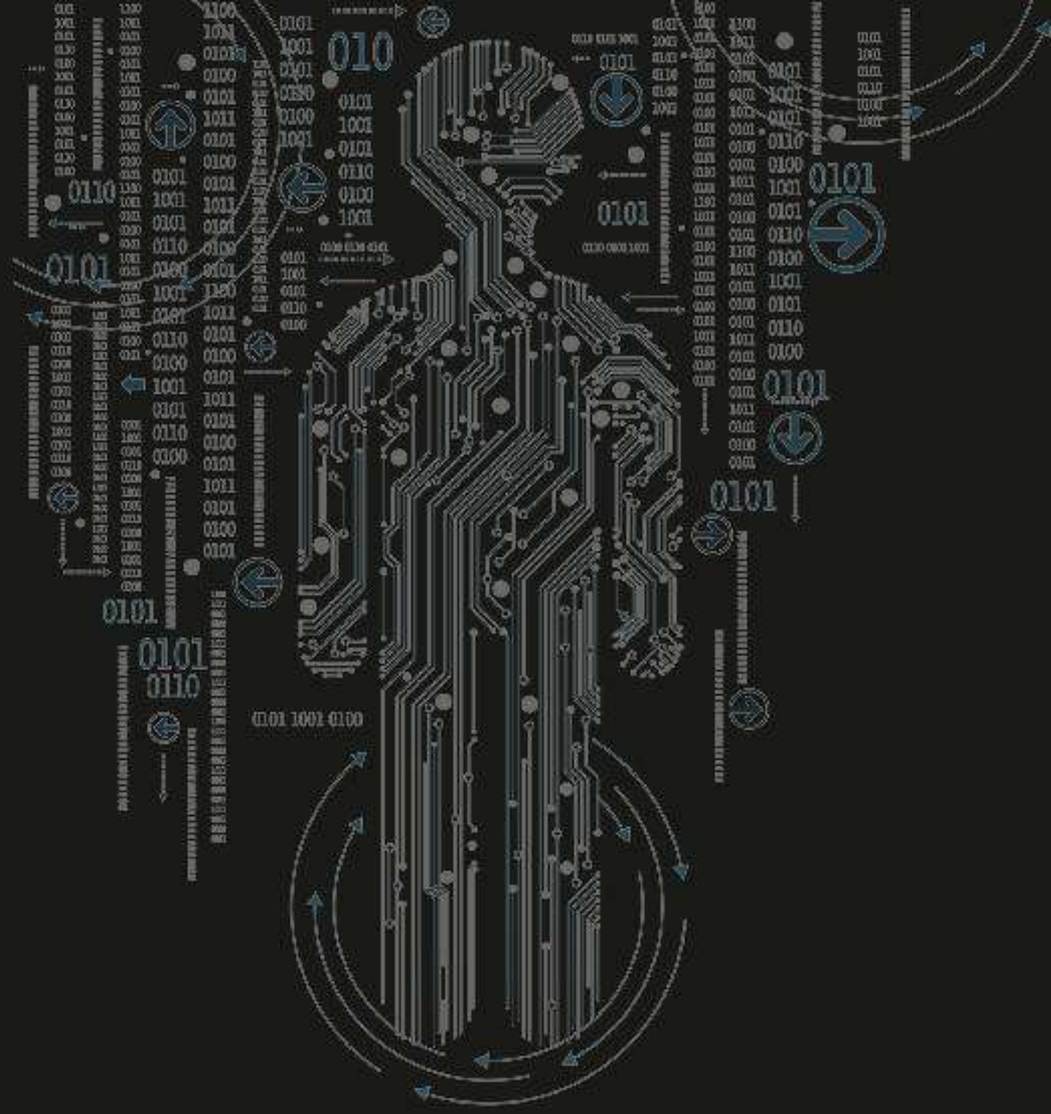
```
Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
    return head;
  q = NULL;
p = head;
while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  ____________________________
  return head;
}
```

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Pointers



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Conditional Control Statements

- if
- if-else
- Nested if-else
- else-if ladder
- Switch

BYJU'S EXAM PREP

"Iterative Statements"

Iteration = Repetition

Iterations or loops are used when we want to execute a statement or block of statements several times.

The repetition of loops is controlled with the help of a test condition. The statements in the loop keep on executing repetitively until the test condition becomes false.

There are the following three types of loops:-

1. while loop
2. do-while loop
3. for loop

BYJU'S EXAM PREP

while loop

It is the fundamental looping statement in C. It is suited for the problems where it is not known in advance that how many times a statement or block of statements will be executed.

The general syntax of while loop is as under:-

```
while(condition)
{
Statement(s);
}
```

Ex:

```
int a=4;
while (a>0)
{ Printf("Hello");
  a = a-1;
}
```

o/p : Hello Hello Hello Hello

a

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

4>0 (T)
3>0 (T)
2>0 (T)
1>0 (T)
0>0 (False)

BYJU'S EXAM PREP

while loop

Entry Control (or) Pre-test loop.

Firstly the condition given with while is evaluated. If the condition evaluates to true then the statements given in the body of the loop are executed. After the execution of all the statements the condition is again checked and if it again evaluates to true then the statements given in the body of the loop are again executed. In this way the statements are executed again and again until the condition given evaluates to false.

For terminating the loop, a termination condition is given in the loop body which makes the condition false at some point of time and as a result of which the loop terminates. If we do not give the termination condition then the loop goes on repeating again and again infinite times. Such loops are referred to as infinite loops.

BYJU'S EXAM PREP

do-while loop ⇒ Post test loop | Exit-control loop.

The do-while statement is also used for looping. It is similar to while loop and is used for the problems where it is not known in advance that how many times the statement or block of statements will be executed.

However, unlike while loop, in case of do-while, firstly the statements inside the loop body are executed and then the condition is evaluated. As a result of which this loop is executed at least once even if the condition is initially false.

After that the loop is repeated until the condition evaluates to false. Since in this loop the condition is tested after the execution of the loop, it is also known as post test loop.

The general syntax of do-while loop is as follows:-

```
do
{
Statement(s);
}while(condition)
```

TRUE

```
int a=4
do
{ printf("Hello");
  a=a-1;
}while(a>0);
```

a 4 3 2 1 0

o/p: Hello
Hello
Hello
Hello

```
int Entered Pin, Actual pin = 7125;
    scanf("%d", & Entered Pin);
```

Minimum one time | At least once.

```
do
{ // dispense cash
} while (Entered Pin == Actual Pin);
```

```
while (Entered Pin == Actual Pin)
{
// dispense Cash
}
```

if wrong pin is Entered first-time itself

do-while : dispense Cash one time

while : Not dispensed Cash.

BYJU'S EXAM PREP

exp1 = initialization executes only 1 time for first time

exp2 = condition

exp3 == Inc/Dec

for loop

The general syntax of for loop consists of three expressions separated by semicolons. It is given as follows:-

```
for(expression1;expression2;expression3)
{
Statement(s);
}
```

All expressions are optional

only 2 semicolons is mandatory.

TRUE

FALSE

Exit

Here the expression1 is the initialization expression,expression2 is the test expression and expression3 is the update expression.

- Expression1 is executed only once when the loop starts and is used to initialize the loop variables.
- Expression2 is a condition and is tested before each iteration of the loop.
- Expression3 is an update expression and is executed each time after the body of the loop is executed.

```
        (1)        (2)        (4)
for ( Exp1 ;  Exp2 ;  Exp3 )
{
    // Body of for loop (3)
}
```

"for" loop is also called Pre-test loop (or) Entry-control loop.

1, 2, 3, 4, 2, 3, 4, 2, 3, 4, 2, 3, 4, ---------- 2 (False)

Ex:1

```
int a;
for(a=1; a<=5; a++)
{ printf("Hello");
}
```

o/p: Hello Hello Hello Hello Hello.

a=1	2<=5 (T)	3<=5	4<=5	5<=5
1<=5 (T)	Hello	Hello	Hello	Hello
Hello	a=3	a=4	a=5	a=6
a=a+1=2				

6<=5 (False)

Ex:2

```
int a=1;
for( ; a<=5; a++)
printf("Hello");
```

o/p: Hello Hello Hello Hello Hello

Ex:3

```
int a;
for(a=1;  ; ++a)
{   if(a<=5)
      printf("Hello");
    else
    break;
}
```

Hello
Hello
Hello
Hello
Hello

Ex: 4

```
int a;
for (a=1; a<=5; _)
{
    Printf("Hello");
    a=a+1;
}
```

o/p: Hello Hello Hello Hello Hello

Ex: 6

a: 1 2 3 4 b: 1 2 3 4

```
int a,b;
for (a=1, b=1; a<3, b<4; a++, b++)
{
    Printf("a=%d b=%d", a,b);
}
```

Exp1: a=1, b=1 Exp2: a<3, b<4

a<3, b<4

1<3, 1<4 (T)
2<3, 2<4 (T)
3<3, 3<4 (T)
4<3, 4<4 (F)

o/p:
a=1 b=1
a=2 b=2
a=3 b=3

Ex: 5

```
int a;
for(a=1; a<=5; a++);
Printf("%d", a);
```

o/p: 6

```
for(a=1; a<=5; a++)
{
    ; // Body
}
Printf("%d", a);
```

a=1
a=2
a=3
a=4
a=5
a=6

$a = \cancel{1}\ \cancel{2}\ 3$ $b = \cancel{1}\ \cancel{2}\ 3$

Ex: 7

```
int a, b;
for (a=1, b=1; b<4, a<3 ; a++, b++)
{ Printf("%d %d", a, b);
}
```

$1 < 4$ $\boxed{1 < 3}$ (T)

$2 < 4$ $\boxed{2 < 3}$ (T)

$3 < 4$ $3 < 3$ (F)

Ends

o/p	
1	1
2	2

BYJU'S EXAM PREP

Jumping Statements

unconditional control statements

- Jump statements are used to transfer the control from one part of the program to another part.
- The various jump statements in C are as under:-

1. break statement : cause control to go out of block.
2. continue statement
3. goto statement

x 12

```
int x;
for(x=1; x<=5; x++)
{ if((x==2) || (x==4))
    break;
  Printf("x=%d", x);
}
```

(1==2) False || (1==4) False

o/p: x=1

```
int x; x=x 2 3 4 5 6
for(x=1; x<=5; x++)
{ if((x==2) || (x==4))
    continue;
  Printf("x=%d", x);
}
```

o/p:
x=1
x=3
x=5

BYJU'S EXAM PREP

break

Break statement is used inside the loops or switch statement. This statement causes an immediate exit from the loop or the switch case block in which it appears.

It can be written as

```
break;
```

BYJU'S EXAM PREP

continue

The continue statement is used inside the body of the loop statement.

when we want to go to the next iteration of the loop after skipping some of the statements of the loop.

The continue statement is written as under:

continue;

The difference between break and continue is that when a break statement is encountered the loop terminates and the control is transferred to the next statement following the loop, but when a continue statement is encountered the loop is not terminated and the control is transferred to the beginning of the loop.

BYJU'S EXAM PREP

goto

It is an unconditional statement and it transfers the control to the another part of the program.

The goto statement is used as under:-

```
goto label;
.................
.................
.................
label:
Statement;
.................
.................
```

Here label is any valid C identifier and is followed by a colon.

```
int a;
if(a>3)
goto xyz;
Pf("Hello");
Pf("Hai");
xyz: Pf("Bye");
```

i/p: a=5
o/p: Bye

a=2 i/p
Hello
Hai } o/p

BYJU'S EXAM PREP

goto

- The goto statement transfers the control to the statement after the label.
- If the label is placed after the goto statement then the control is transferred forward and it is known as forward jump.
- If the label is placed before the goto statement then the control is transferred backward and the jump is called backward jump.
- In the forward jump all the statements between the goto statement and the label are skipped .
- In case of backward jump all the statement between the goto statement and the label are executed.
- There should always be a statement after any label.

BYJU'S EXAM PREP

Q1. Consider the following C code. Assume that unsigned long int type length is 64 bits.

```
unsigned long int fun(unsigned long int n){
    unsigned long int i, j = 0, sum = 0;
    for (i = n; i > 1; i = i/2) j++;
    for ( ; j > 1; j = j/2) sum++;
    return(sum);
}
```

$i = 2^{40}$ $\quad i = 2^{39}$ $\quad i = 2^{38}$ ... $i = 2^{1}$ $\quad i = 2^{0}$

$2^{40} > 1$ $\quad 2^{39} > 1$ $\quad 2^{38} > 1$ --- $2^{1} > 1$ $\quad 2^{0} > 1$ (False)

$j = \cancel{0}\,1$ $\quad j = 2$ $\quad j = 3$ --- $j = 40$

$j = 40 > 1$ $\quad 40/2 = 20$ $\quad 10$ $\quad 5$ $\quad 5/2 \Rightarrow 2$ $\quad 1 > 1$ (False)

$Sum = \cancel{0}\,1$ $\quad 2$ $\quad 3$ $\quad 4$ $\quad \boxed{5}$

The value returned when we call fun with the input 2^{40} is _____ .

[GATE 2018]

Sum = 5

BYJU'S EXAM PREP

* /
+

ANS: 10

Q2. Consider the following C program:

```
# include<stdio.h>
int main ( )
{
  int i, j, k = 0;
  j = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
  k -= - -j;
  for (i = 0; i < 5: i++)
  {
    switch (i + k)      {
      case 1:
      case 2: printf ("\ n%d", i+k);
      case 3: printf ("\n%d", i+k);
      default: printf ("\n%d", i+k);      }
  }
  return 0;
}
```

$2\times3=6$

$6/4=1$

$2.0/5=0.4$

$8/5=1$

$1+0.4+1$
$=2.4$

$j=\cancel{2}\ 1$

$k = k - --j; = 0-1=-1 \Rightarrow k=-1$

$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
$i+k = 0+(-1)$	$1+(-1)$	$2+(-1)$	$3+(-1)$	$4+(-1)$
$=-1$	$=0$	$=1$	$=2$	$=3$
default (1)	default (2)	Case 2 (3), Case 3 (4), default (5)	Case 2 (6), Case 3 (7), default (8)	Case 3 (9), default (10)

The number of times printf statement is executed is _____.

[GATE 2015]

BYJU'S EXAM PREP

Q3. What will be the output of the following C program segment?

```
char inChar = 'A' ;
switch ( inChar ) {
case 'A' : printf ("Choice A\ n") ;
case 'B' :
case 'C' : printf ("Choice B") ;
case 'D' :
case 'E' :
default : printf ( " No Choice" ) ; }
```

[GATE 2012]

A. No Choice

B. Choice A

C. Choice A,
Choice B No Choice

D. Program gives no output as it is erroneous

BYJU'S EXAM PREP

Home work

Q4. Consider the following C program

```
main()
{
  int x, y, m, n;
  scanf ("%d %d", &x, &y);
  /* x > 0 and y > 0 */
  m = x; n = y;
  while (m != n)
  {
    if(m>n)
      m = m - n;
    else
      n = n - m;
  }
  printf("%d", n);
}
```

The program computes

[GATE 2004]

A. x + y using repeated subtraction.

B. x mod y using repeated subtraction.

C. The greatest common divisor of x & y.

D. The least common multiple of x & y.

BYJU'S EXAM PREP

Pointer

To be Contd... Tomorrow @ 12PM extra class.
[Pointers & functions]

- Pointer is a variable that holds address of another data item.
- A pointer can hold address of a variable or an array or a function or a structure or a file or another pointer also.
- Example:

A) Tomorrow

B) Now

BYJU'S EXAM PREP

Pointer Declaration and Initialization

- The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.
- Example:

```
int *a;//pointer to int

char *c;//pointer to char
```

BYJU'S EXAM PREP

Pointer Data Type

- As pointers hold addresses, address cannot be negative and fractional, So the data type of pointer will always be unsigned integer.
- While declaring a pointer, data type describes, what kind of data address, a pointer should hold.

BYJU'S EXAM PREP

Pointer Size

- As pointer holds address and as address is always unsigned integer type, size of pointer will always be integer size, while the integer size is compiler dependent.
- Example:

BYJU'S EXAM PREP

Pointer Arithmetic

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- Increment
- Decrement
- Addition
- Subtraction
- Comparison

BYJU'S EXAM PREP

Types Of Pointers

There are various types of pointers, that include:

- Void pointer
- Null Pointer
- Dangling Pointer
- Near Pointer
- Far Pointer
- Array Pointer
- Function Pointer
- Structure Pointer
- Pointer to Pointer etc.

BYJU'S EXAM PREP

Null Pointer

A Null Pointer is a pointer that does not point to any memory location. It stores the base address of the segment. The null pointer basically stores the Null value while void is the type of the pointer.

A null pointer is a special reserved value which is defined in a stddef header file. Here, Null means that the pointer is referring to the 0th memory location.

If we do not have any address which is to be assigned to the pointer, then it is known as a null pointer. When a NULL value is assigned to the pointer, then it is considered as a Null pointer.

Following are the applications of a Null pointer:

- It is used to initialize 0 pointer variable when the pointer does not point to a valid memory address.
- It is used to perform error handling with pointers before dereferencing the pointers.
- It is passed as a function argument and to return from a function when we do not want to pass the actual memory address.

BYJU'S EXAM PREP

Void Pointer

if we declare the int pointer, then this int pointer cannot point to the float variable or some other type of variable, i.e., it can point to only int type variable.

To overcome this problem, we use a pointer to void. A pointer to void means a generic pointer that can point to any data type.

We can assign the address of any data type to the void pointer, and a void pointer can be assigned to any type of the pointer without performing any explicit typecasting.

Syntax of void pointer

void *pointer name;

Declaration of the void pointer is given below:

void *ptr;

BYJU'S EXAM PREP

Pointer to Pointer

As we know that, a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable.

However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer).

The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.

The syntax of declaring a double pointer is given below.

```
int **p; // pointer to a pointer which is pointing to an integer.
```

BYJU'S EXAM PREP

Dangling Pointer

A pointer with an invalid address is known as a dangling pointer in C.

Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer.

In this case, the pointer is pointing to the memory, which is de-allocated. The dangling pointer can point to the memory, which contains either the program code or the code of the operating system.

BYJU'S EXAM PREP

Const Pointer

A constant pointer in C cannot change the address of the variable to which it is pointing, i.e., the address will remain constant. Therefore, we can say that if a constant pointer is pointing to some variable, then it cannot point to any other variable.

Syntax of Constant Pointer

<type of pointer> *const <name of pointer>;

The Declaration of a constant pointer is given below:

int *const ptr;

BYJU'S EXAM PREP

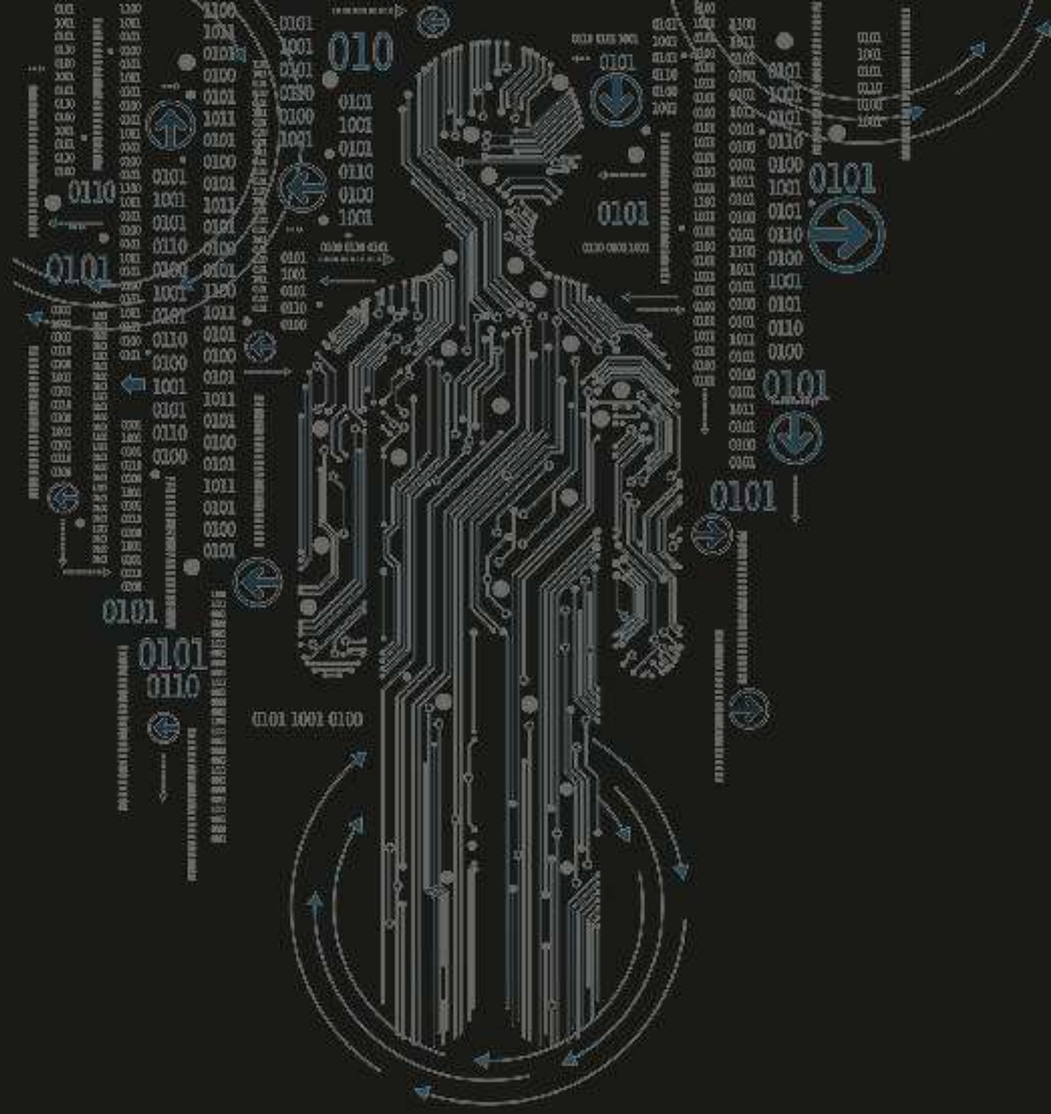
Examples

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Recursion



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

- Pointers : variable, holds address of any data
  - Pointer
  - Structure
  - variable
  - array
  - Function
  - Constant
  - File

- Data Type : Unsigned Integer
- Size of Pointer : Integer size – Compiler Dependant
- Types of Pointers – NULL, Void, Array Pointer, Function Pointer, Structure Pointer, Pointer to Pointer, Constant Pointer
- Pointer Arithmetic
- Function Definition, Types of functions
  - System-defined
  - User-defined.

BYJU'S EXAM PREP

Function Properties

Every 'C' function is comprising of following properties:

- **Function Name**
- **Arguments / Parameters** : inputs
- **Return Type** : output type [Return value datatype]
- **Body of the function** : Code Inside function block

```
Returntype  Name (arguments)
{
    // Body
}
```

Ex:

```
int xyz (int a)
{
    a=a*2;
    Printf("%d", a);
}  return a;
```

① Name ② Return type ③ Body ④ Arguments (optional)

Function Definition

```
②          ①      ④
Returntype Name(arguments*)
{
    // Body ③
}
```

Ex:

```
int A(int x, int y)
{
    int z;
    z = x + x/y;
    return z;
}
```

Definition

Body

Function Declaration

```
②          ①      ④
Returntype Name(arguments type*);
```

```
int A(int, int);
```

```
Name(arguments*); // Calling
```

```
A(5, 7);
```

```
main(      )
{
    - - -
    - -   // Body is
    - -     developed by
    - -       User.
}
```

⇒ User is defining body of main function. So, main( ) is User-defined function.

```
Printf(     )
{ - -
}

Scanf(   )
{ - -
}

getch(  )
{ - -
}
```

User will not develop body. User just call those system-defined functions.

BYJU'S EXAM PREP

Function Declaration

Every 'C' program execution starts from main() function.

When any 'c' function is invoked / called before it is defined, then it must be declared before.

// Prototype declaration.

Syntax:

Return _type Name(arguments type);

Example:

```
void fun(void);

int fun(int,int);
```

BYJU'S EXAM PREP

Function Definition

Syntax:

```
Return_type Name(Arguments)
{
-------
-------
Body
------
}
```

Note: Arguments are always optional

// Defining Add function

```
// Callee (or) called function
void Add(int x, int y)
{
    int z;
    z = x+y;
    Printf("%d", z*z);
}

void main( )   // Caller (or) Calling fun
{
    Add(5, 6);   // Calling Add function
}
```

→ Prototype declaration

```
void Add(int, int);

void main( )   // Caller
{
    Add(5, 6);   // Calling
}

// defining
void Add(int x, int y)   // Callee
{
    int z;
    z = x+y;
    Printf("%d", z*z);
}
```

BYJU'S EXAM PREP

Caller, Callee functions

A function which calls another function is said to be caller or calling function.

A function which is called by another function is said to be callee or called function.

Example :

```
void main() //Caller (or) Calling function
{
Add(5,6);
}
void Add(int x, int y) // Callee (or) Called function.
{
printf("Sum is %d", x+y);
}
```

```
void B(void);
void A( )
{
  Printf("Hello");
  B();  //Calling 'B'
  Printf("Hey");
}
void C(void);
void B( )
{
  Printf("Hai");
  C();
  Printf("chalo");
}
```

```
void main( )
{
  Printf("welcome");
  A();  //Calling 'A'
  Printf("Bye");
}
void C( )
{
  Printf("Thank you");
}
```

o/p: welcome Hello Hai Thank you chalo Hey Bye

BYJU'S EXAM PREP

Function Prototypes

Functions can be implemented in either of 4 different ways:

- Functions with arguments and with return values
- Functions with arguments and without return values
- Functions without arguments and with return values
- Functions without arguments and without return values

Arguments	R.values
✗	✗
✗	✓
✓	✗
✓	✓

BYJU'S EXAM PREP

Example

```
void A( )//Callee
{
    Printf("Hello");
}
Void main( ) //Caller
{
    A();
}
```

Function without arguments without return value.

```
int A( ) //Callee
{ int x = 5;
    return x*x*x;
}
Void main( ) //Caller
{ int result;
  result = A( );
  Printf("%d", result);
}
```

Functions without arguments, but with return value

x, y : Arguments while definition (Formal)
a, b : Arguments while Calling (Actual)

BYJU'S EXAM PREP

```
//Callee
void Add (int x, int y)
                    // Formal arguments
{
    int z;
    z = x * y;
    Printf("%d", z);
}
void main( ) //Caller
{ int a=5, b=7;
    Add (a, b);     // Actual arguments
}
```

Functions with arguments, without return value

```
//Callee
int Add (int x, int y)
{ int z;
    z = x * y;
    return z;
}
void main( ) // Caller
{ int a=5, b=7;
  Printf("%d", Add(a,b));
}
```

Functions with arguments, with return value

32-bit Processor

BYJU'S EXAM PREP

Storage Classes

A storage class defines:

- Default Value
- Memory Location
- Scope and
- Lifetime of a variable

There are 4 types of storage classes : Automatic, Register, Static and External

int x;

4 Bytes Memory allocated where? (1) Memory location

Printf("%d", x); (2) default value

What Value is Printed?

Block A { int x; }

Block B { use 'x'. }

Till what limit variable can be accessed? (3) Scope.

(4) lifetime

BYJU'S EXAM PREP

Register: For frequently used variable.

int a; == auto int a;

Storage Classes

keyword

Syntax : Storage class datatype variable;

Ex – static int x;

auto char y;

register float z;

extern double d;

Storage class	Memory Location	Default Value	Scope	Lifetime
auto Automatic (default)	RAM	Garbage value	Block scope	within the block
register Register	CPU registers	Garbage value	Block scope	within block
static Static	RAM	ZERO	Block scope	Throughout the Program
extern External	RAM	ZERO	File scope	Throughout program

BYJU'S EXAM PREP

Examples

```
void A( ) {
    auto int a=1;
    a=a+1;
    Printf("%d", a);
}
void main( )
{ int i;
  for (i=1; i<5; i++)
    A( );
}
```

a: ~~12~~ ~~a: 12~~ ~~a: 12~~

a: 12

i=1 1<5 (T)
A(); → 2

i=2 2<5 (T)
A(); → 2

i=3 3<5 (T)
A(); → 2

i=4 4<5 (T)
→ 2

i=5, 5<5 (F)

O/P:- 2 2 2 2

NOTE: Static & extern members are initialized only once.

```
void A( )
{
  static int a = 1;
    a=a+1;
    Printf("%d", a);
}

void main( )
{ int i;
  for(i=1; i<5; i++)
    A();
}
```

a: 1 → 2 → 3 → 4 → 5

i=1 1<5 (T)

A() → Prints 2

i=2 2<5 (T)

A() → Prints 3

i=3 3<5 (T)

A() → Prints 4

i=4 4<5 (T)

A() → Prints 5

i=5 5<5 (F)

O/P: 2 3 4 5

BYJU'S EXAM PREP

Recursion

In Programming, When one or multiple statements need to be executed repeatedly, then it is possible using either control statements or recursion.

The Process of calling a function by itself is known as recursion and such function is said to be recursive function.

```
void A(int);
void main( )
{
    A(5);
}
```

// Caller for A

```
void A(int x) // Callee
{
    if(x==0)
        return;
    else
    {
        Printf("%d", x);
        A(x-1);
    }
}
```

// A Calling A itself.

= A is recursive function.

BYJU'S EXAM PREP

Recursion

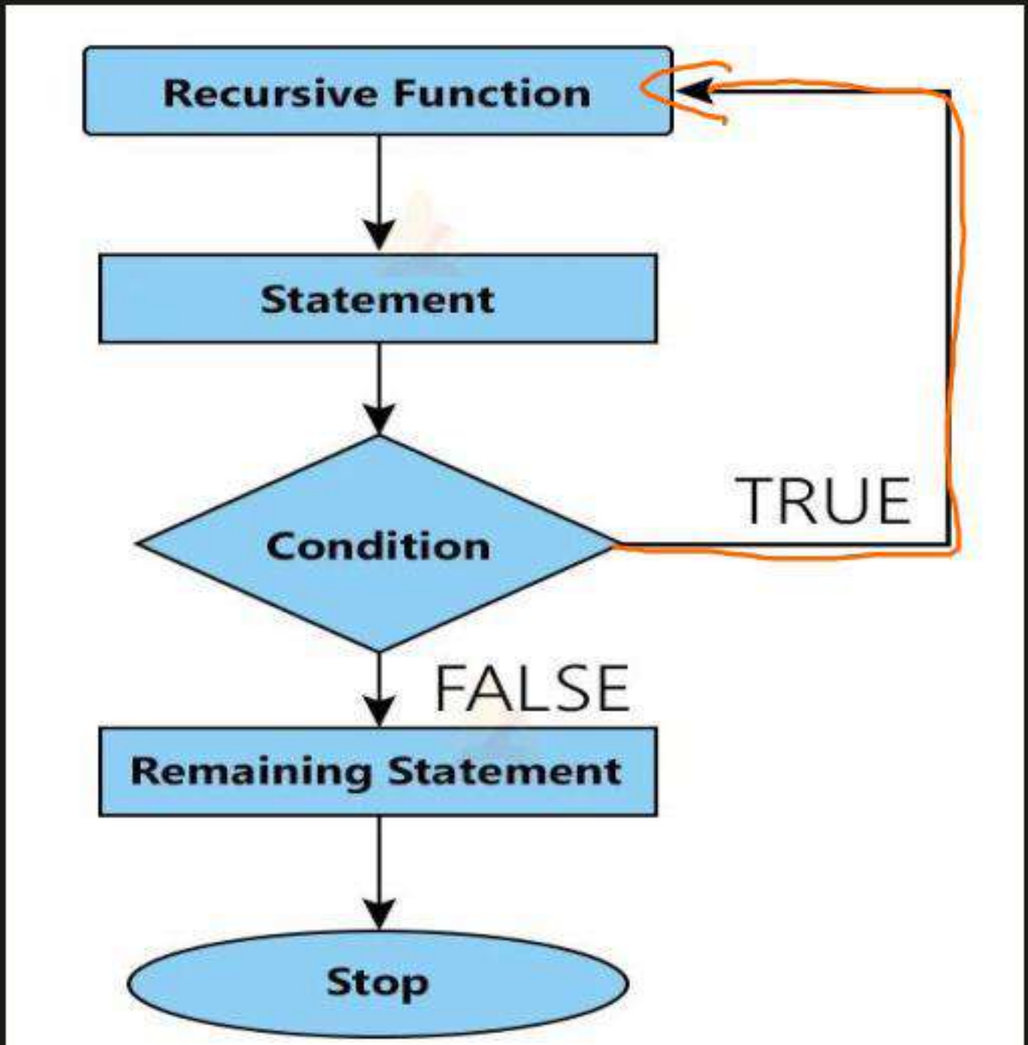
While recursive implementation, the condition where recursive calls stop is said to be base condition or base case.

If base case is missing in recursive implementation, then it may cause infinite execution.

Syntax:

```
Function(arguments)
{
……
Function(arguments);
}
```

Recursion Flow chart



Flowchart showing Recursion

BYJU'S EXAM PREP

Types Of Recursion

There are two types of recursion present in the C programming language.

1. Direct Recursion : If a function calls itself directly then the function is known as direct recursive function.
2. Indirect Recursion: When a function is mutually called by another function in a circular manner, the function is called an indirect recursion function.

BYJU'S EXAM PREP

Direct Recursion

Ex:

```
int fib(int num)
{
    if (num==1 || num==2)
        return 1;
    else
        return (fib(num-1)+fib(num-2));
}
```

Types Of Direct Recursion

To be contd...

Direct Recursion Can be implemented in either of following ways:

- Head Recursion
- Tail Recursion
- Nested Recursion
- Tree Recursion

BYJU'S EXAM PREP

Head Recursion

When the recursive call is the first or early statement in recursive function, then it is called head recursion.

Example:

```
int fun(int x)
{
if(x==0) return;
else{
fun(x-1);
printf("x=%d",x);
}
}
```

Tail Recursion

When the recursive call is the last or later statement in recursive function, then it is called Tail recursion.

Example:

```
int fun(int x)
{
if(x==0) return;
else{
printf("x=%d",x);
fun(x-1);
}
}
```

BYJU'S EXAM PREP

Nested Recursion

When the recursive call is made inside another recursive call and so on, then it is said to be nested recursion.

Example:

```
int fun(int x)
{
if(x==0) return;
else{
return fun(fun(x-1));
}
}
```

Tree Recursion

When the recursive call is made multiple times within the same function, then it is said to be Tree recursion.

Example:

```
int fun(int x)
{
if(x==0) return;
else{
return fun(x-1)+fun(x-3);
}
}
```

BYJU'S EXAM PREP

Indirect Recursion

When the recursive call is made multiple times within the same function, then it is said to be Tree recursion.

```
int first_function(int num)
{
    if (num<=2)
        return 2;
    else
        return new_function(num);
}
int new_function(int num)
{
    return first_function(num);
}
```

BYJU'S EXAM PREP

GATE 2011

Consider the following recursive C function that takes two arguments:

```
unsigned int foo (unsigned int n, unsigned int r) {
  if (n > 0) return((n % r) + foo(n/r, r));
  else return 0;
}
```

What is the return value of the function foo when it is called as foo (513, 2)?

BYJU'S EXAM PREP

GATE 1998

What value would the following function return for the input x = 95?

```
Function fun (x:integer):integer;
Begin
    If x > 100 then fun : x – 10
    Else fun : fun(fun (x + 11))
End;
```

BYJU'S EXAM PREP

GATE 2018

Consider the following C program:

```
#include < stdio.h >
  int counter = 0;
  int calc (int a, int b) {
  int c;
  counter++;
  if (b==3) return (a*a*a);
  else {
    c = calc(a, b/3);
    return (c*c*c);
 }
}
int main (){
  calc(4, 81);
  printf ("%d", counter);
}
```

The output of this program is _____.

BYJU'S EXAM PREP

GATE 2005

Consider the following C program:

```
void foo (int n, int sum) {
  int k = 0, j = 0;
  if(n==0) return;
  k=n%10; j = n /10;
  sum = sum + k;
  foo(j, sum);
  printf("%d",k);
}
int main () {
  int a = 2048, sum = 0;
  foo(a, sum);
  printf("%d\n", sum);
}
```

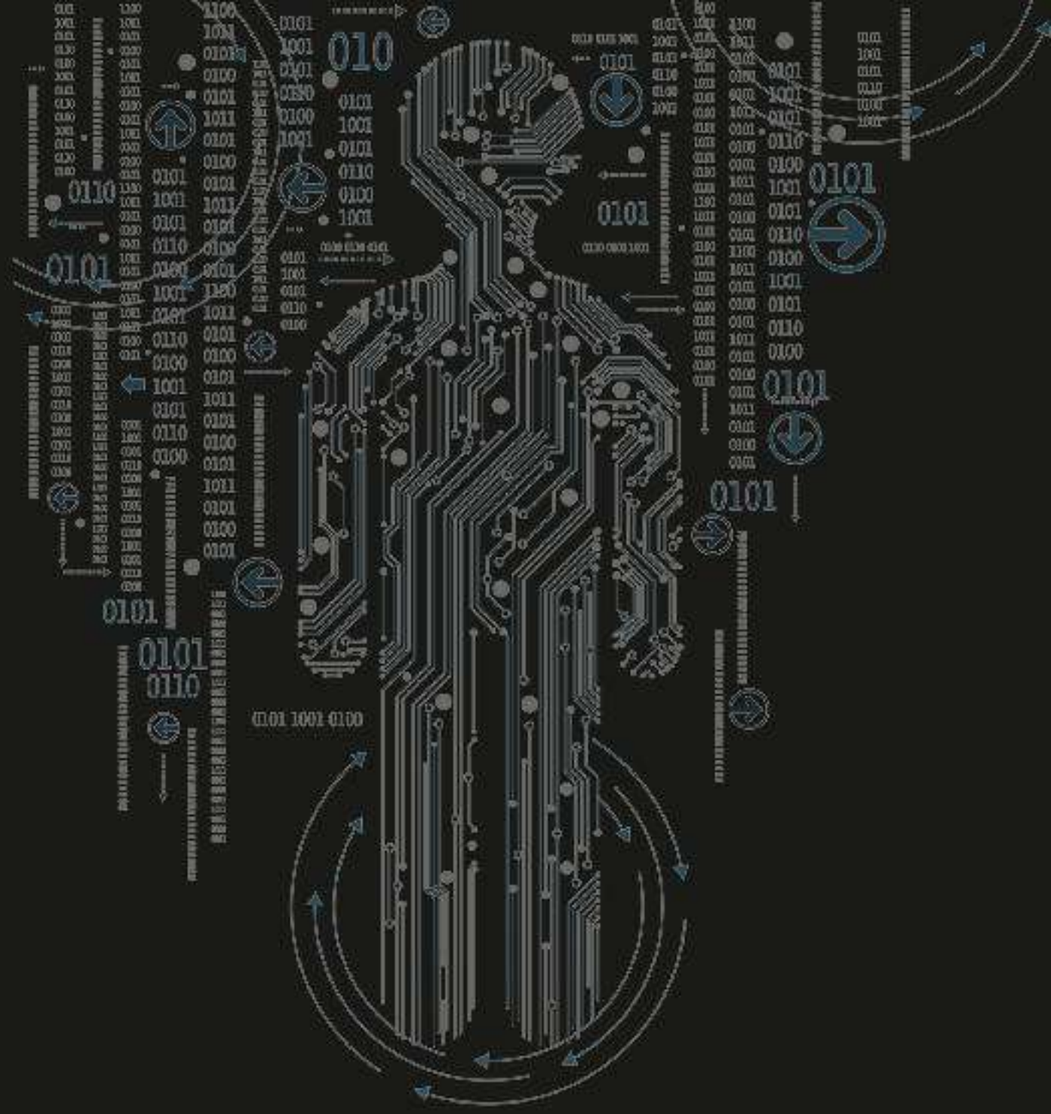
What does the above program print?

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Functions, Pointers



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

- Iterative Control Statements (loops)
  - do-while
  - while
  - for

- Unconditional | Jumping statements
  - break
  - goto
  - continue
  - return

16-bit Processor

```
int a = 7;
char b = '@';
float c = 5.271;
double d = 0.012467;
```

address – Hexadecimal Notation

In Memory Each content is identified with it's address (unique)

a – Name – b

7	– value –	@

2 Bytes – Size – 1 Byte

A_1 – Address – A_{100}

c – Name – d

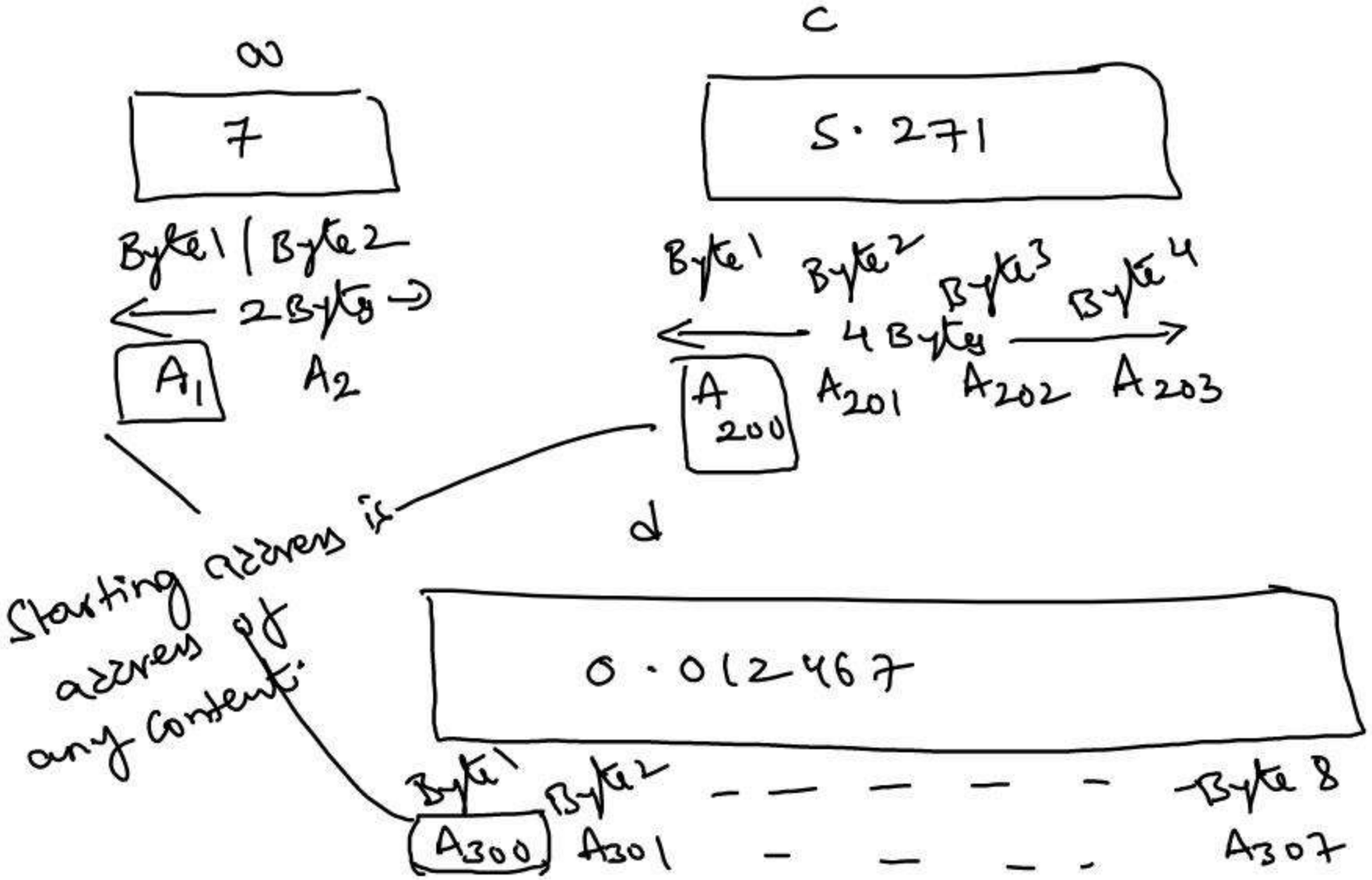
5.271	– value –	0.012467

4 Bytes – Size – 8 Bytes

A_{200} – Address – A_{300}

In Primary | Main | RAM

Memory is Byte-addressable. [Each Byte will have unique address]



BYJU'S EXAM PREP

Pointer

int x[5] = {10, 20, 15, 17, 21};

10	20	15	17	21

Starting address / Base address → A_{100} ← 10 Bytes →

Pointer to array.

int (*q)[5]

q = &x;

- Pointer is a variable that holds address of another data item.
- A pointer can hold address of a variable or an array or a function or a structure or a file or another pointer also.
- Example:

int a = 7;

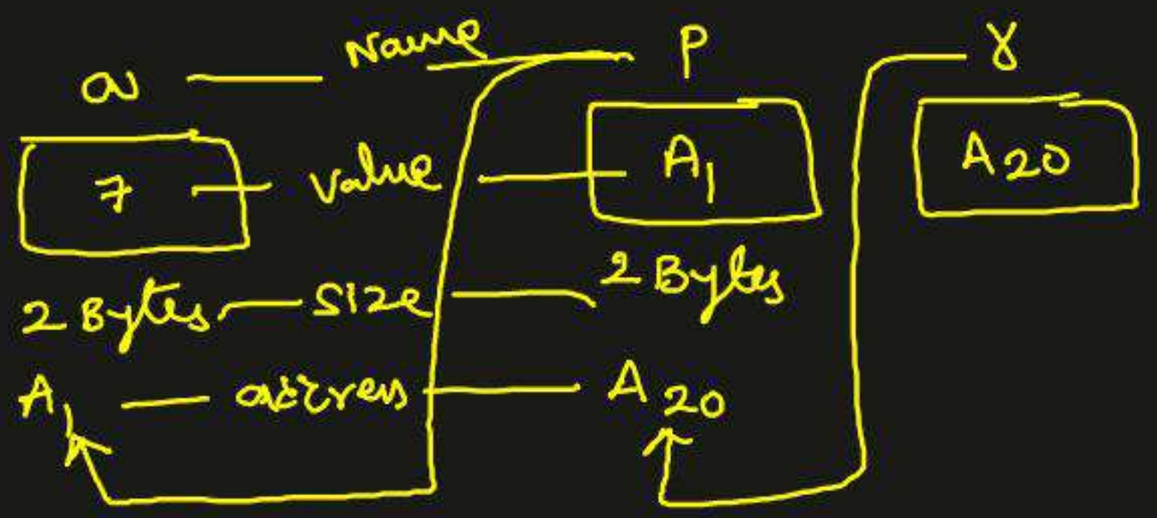
16-bit Processor

int * p;

p = &a;

int **r;

r = &p;



// r is Pointer to Pointer

BYJU'S EXAM PREP

Pointer Declaration and Initialization

- The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.
- Example:

```
int *a;//pointer to int
char *c;//pointer to char
```

float *d; // Pointer to float

* = Indirection operator
[OR]
dereference operator.

BYJU'S EXAM PREP

Pointer Data Type == Unsigned Integer.

+ve only whole number

STUDENTS
CSE ECE MECH CIVIL

- As pointers hold addresses, address cannot be negative and fractional, So the data type of pointer will always be unsigned integer.
- While declaring a pointer, data type describes, what kind of data address, a pointer should hold.

int a=5, *p; ⇒ p is Pointer of type Integer that Points to integer

char b='a', *q; ⇒ q is Pointer of type Integer that Points to characters

float c=1.23, *r; ⇒ r is Pointer of type Integer that Points to float

p=&a;
q=&b;
r=&c;
} valid

p=&b	q=&a	r=&a
p=&c	q=&c	r=&b

Invalid
// Type Mismatch Error.

BYJU'S EXAM PREP

Pointer Size == Integer size

- As pointer holds address and as address is always unsigned integer type, size of pointer will always be integer size, while the integer size is compiler dependent.
- Example: 16-bit Processor

```
int a=7, *p;
char b='f', *q;
float c=1.12, *r;
p=&a;
q=&b;
r=&c;
```

a: 7 — A1, 2 Bytes

b: f — A10, 1 Byte

c: 1.12 — A40, 4 Bytes

p: A1 — 2 Bytes

q: A10 — 2 Bytes

r: A40 — 2 Bytes

BYJU'S EXAM PREP

Pointer Arithmetic

int a; // Normal Variable
// on 'a' all unary, binary, Ternary operations are Possible

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

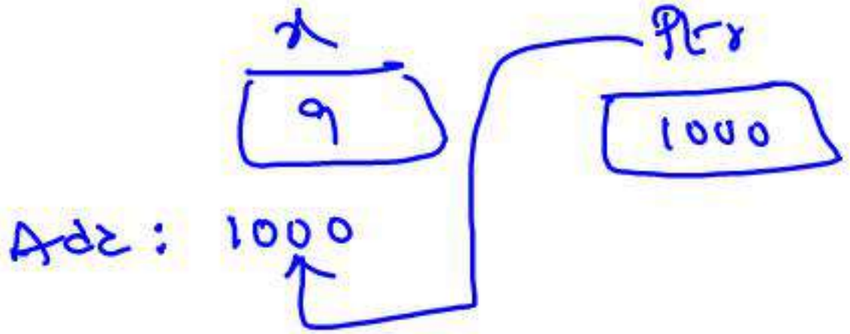
- Increment — P++ ✓
- Decrement — P-- ✓
- Addition — P+Integer Constant
- Subtraction — P− Integer Constant
- Comparison — Type Comparisions

int *p;
P = NULL; (Assignment of NULL)

16-bit Processor => int = 2 Byte

Let, int x = 9, *ptr;

ptr = &x;



ptr++; 1002

ptr--; 998

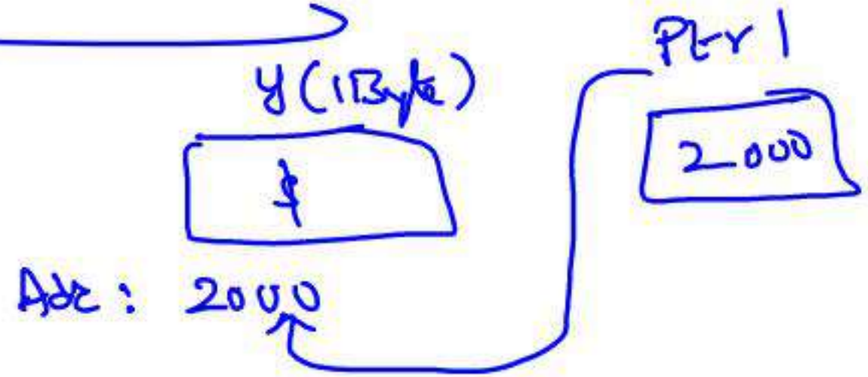
ptr+4
1000 + 4*2
= 1000+8
= 1008

---

char y = 'f', *ptr1;

ptr1 = &y;

ptr1++; 2001



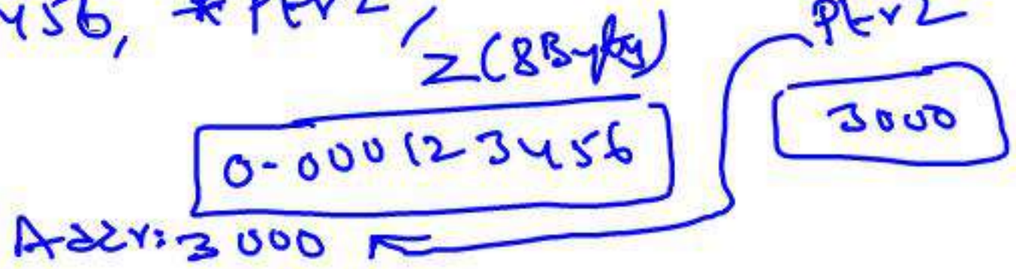
ptr1+6
= 2000 + 6(1)
= 2006

---

double z = 0.000123456, *ptr2;

ptr2 = &z;

ptr2++; 3008



ptr2+9
= 3000 + 9(8)
= 3072

Pointer + i == address + i (size of an element of type Pointer)

BYJU'S EXAM PREP

Types Of Pointers

Generic Pointer

There are various types of pointers, that include:

- Void pointer : A Pointer which can hold any type of data address.
- Null Pointer : A Pointer which Points to 'NULL' value int *p = NULL;
- Dangling Pointer : A Pointer which Points to an invalid location.
- Near Pointer
- Far Pointer
- Array Pointer
- Function Pointer
- Structure Pointer
- Pointer to Pointer etc.

```
int a;
char b;
float c;
void *x;
```

```
x = &a;
x = &b;
x = &c;
```

```
char *s;
char *p;
p = &s;
free(s);
```

Dangling Pointer.

↳ It Erase Memory

p

'p' still Points to A10

BYJU'S EXAM PREP

Null Pointer

A Null Pointer is a pointer that does not point to any memory location. It stores the base address of the segment. The null pointer basically stores the Null value while void is the type of the pointer.

A null pointer is a special reserved value which is defined in a stddef header file. Here, Null means that the pointer is referring to the 0th memory location.

If we do not have any address which is to be assigned to the pointer, then it is known as a null pointer. When a NULL value is assigned to the pointer, then it is considered as a Null pointer.

Following are the applications of a Null pointer:

- It is used to initialize 0 pointer variable when the pointer does not point to a valid memory address.
- It is used to perform error handling with pointers before dereferencing the pointers.
- It is passed as a function argument and to return from a function when we do not want to pass the actual memory address.

BYJU'S EXAM PREP

Void Pointer

if we declare the int pointer, then this int pointer cannot point to the float variable or some other type of variable, i.e., it can point to only int type variable.

To overcome this problem, we use a pointer to void. A pointer to void means a generic pointer that can point to any data type.

We can assign the address of any data type to the void pointer, and a void pointer can be assigned to any type of the pointer without performing any explicit typecasting.

Syntax of void pointer

void *pointer name;

Declaration of the void pointer is given below:

void *ptr;

BYJU'S EXAM PREP

Pointer to Pointer

As we know that, a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable.

However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer).

The first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.

The syntax of declaring a double pointer is given below.

```
int **p; // pointer to a pointer which is pointing to an integer.
```

BYJU'S EXAM PREP

Dangling Pointer

A pointer with an invalid address is known as a dangling pointer in C.

Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer.

In this case, the pointer is pointing to the memory, which is de-allocated. The dangling pointer can point to the memory, which contains either the program code or the code of the operating system.

BYJU'S EXAM PREP

Keyword

Const Pointer

A constant pointer in C cannot change the address of the variable to which it is pointing, i.e., the address will remain constant. Therefore, we can say that if a constant pointer is pointing to some variable, then it cannot point to any other variable.

Syntax of Constant Pointer

<type of pointer> *const <name of pointer>;

The Declaration of a constant pointer is given below:

int *const ptr; a = 7,

ptr = &a;

ptr = ptr1; (or) ptr = &b; // Error Constant Pointer Cannot be Modified.

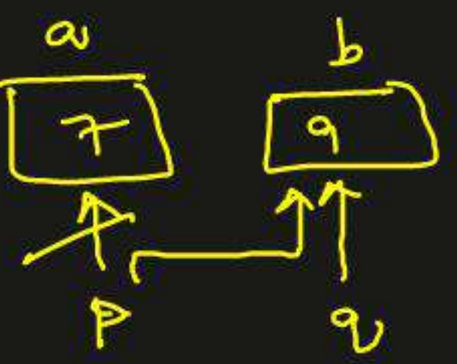
int a=7, b=9;

int *p, *q;

p = &a;

q = &b;

p = q;

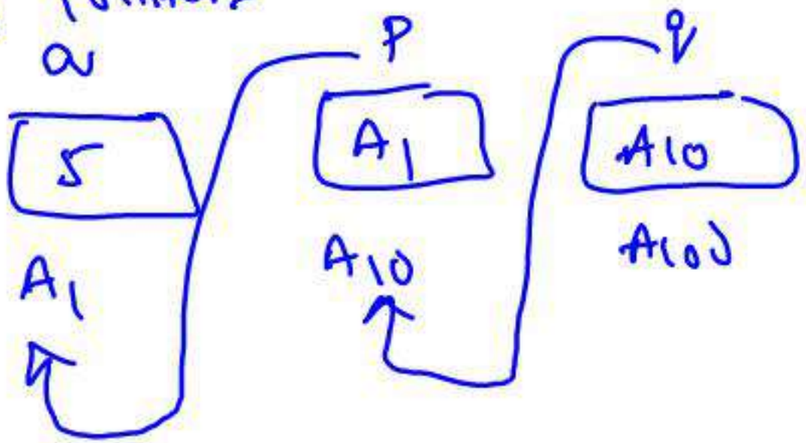


* Is to indicate compiler that, they are pointers.

$\boxed{\text{int } a=5, *p, **q;}$

p = &a;

q = &p;



a=5 Printf("a = %d", a);

a=5 Printf("a = %d", *p);

a=5 Printf("a = %d", **q);

$\boxed{* = \text{value at address}}$

*p = value at address p

= value at address A_1 = 5

**q = *(value at address A10)

= *(A_1)

= value at address A_1 = 5

// Here "a" value is accessed indirectly through "*". Hence * is called indirection operator.

BYJU'S EXAM PREP

FUNCTIONS

'C' is Procedure - Oriented

'C' Language is procedure-oriented language, means that, In 'C', A big task is sub divided into small sub tasks and every sub task is implemented individually.

The code of each sub task can be called as Procedure.

Procedure

Sub Task1 { }

Procedure

{ } Sub Task2

Task:

Ex: Accept a number and Print is it Even or odd { } it's table, Sub Task3 it's factorial { } Sub Task 4 is it Prime or not

Sample i/p: 5

Sample o/p: 5 is odd

5 is Prime

5x1=5

5x2=10

5x10=50

5 factorial is 120.

BYJU'S EXAM PREP

Function

Algorithm Term — Program Term.

Procedure == Function

A single statement or group of statements, that perform specific sub task is called as Function.

There exists 2 types of functions:

1. System-defined functions (or) Pre-defined (or) Library functions
2. User-defined functions

Ex: main()

Ex: Printf(), Scanf(), clrscr(), getchar(), putchar(), strlen(), sizeof, malloc(), calloc(), ----

BYJU'S EXAM PREP

Function Properties

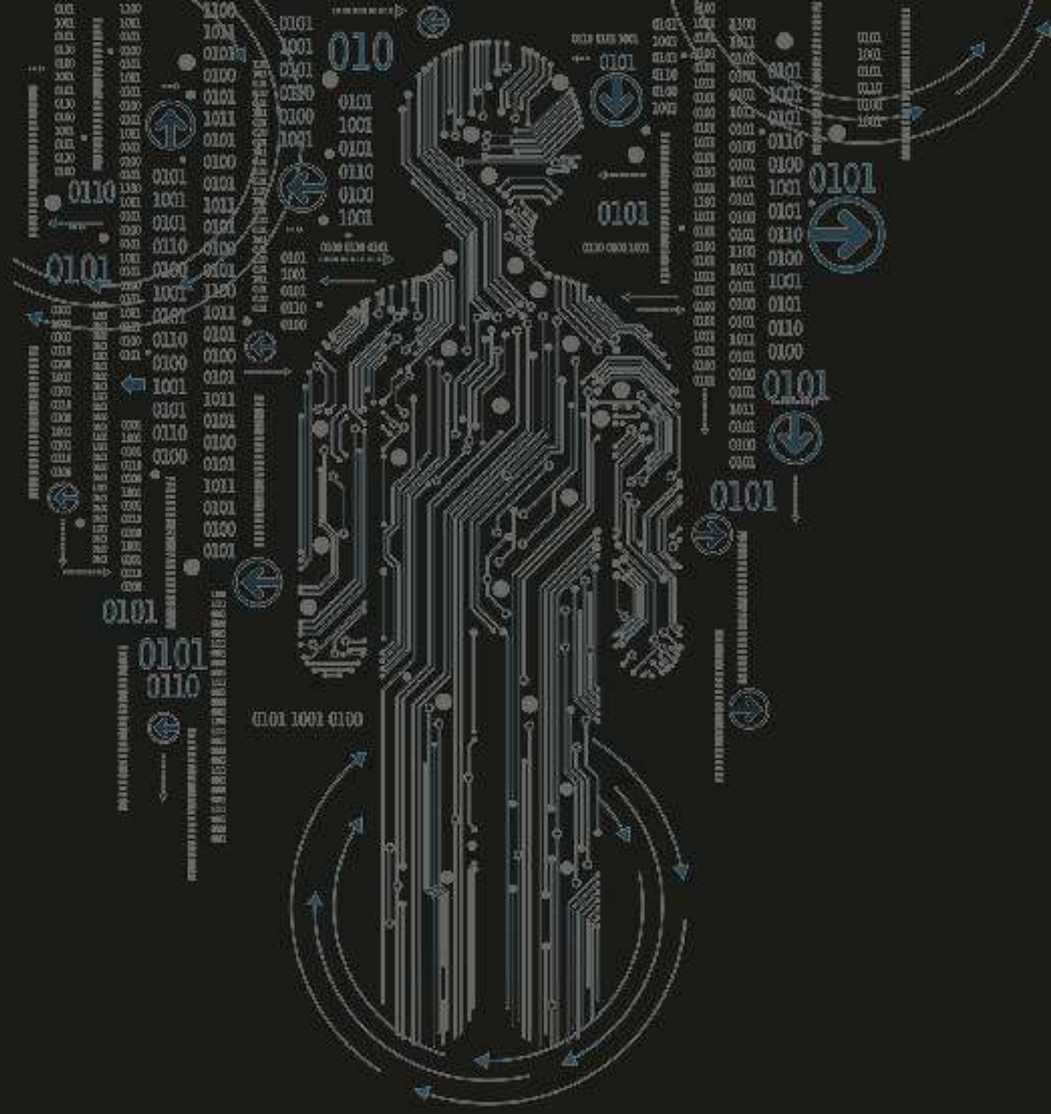
Every 'C' function is comprising of following properties:

- **Function Name**
- **Arguments / Parameters**
- **Return Type**
- **Body of the function**

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Problem Solving @ 'C' Programming



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Self – Referential Structure ⟹ Linked Lists

A structure whose member is variable of itself is called as self referential structure.

Syntax:

```
struct Structure{

.....

struct Structure *variable;

};
```

Ex.

```
Struct ABC
{
    int a;
    Struct ABC *b;
} *v1, *v2;
```

members

Variables

→ Self referential Structure

Members: a, b

Variables: v_1, v_2

- Variables must be Pointer variables, otherwise it leads to infinite Memory allocation → segmentation fault.

What is o/p ?

Example - 1

```c
#include <stdio.h>
struct S {
int S;
};
int main(void) {
struct S S;
S.S = sizeof(struct S) / sizeof(S);
printf("%d",S.S);
return 0;
}
```

"Member S"

Structure S

Variable S

Structure S

Variable S

Structure S

Variable S

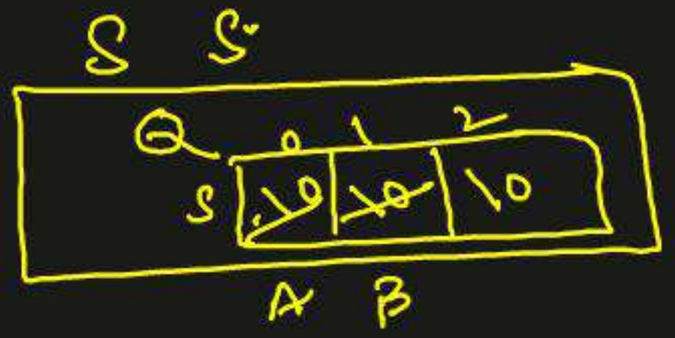
S

$\frac{2 \text{ Bytes}}{2 \text{ Bytes}}$ $\frac{4 \text{ Bytes}}{4 \text{ Bytes}}$ $= 1$

What is o/p ?

Example - 2

```
#include <stdio.h>
#include <string.h>
struct Q {
char S[3];
};
struct S {
struct Q Q;
};
int main(void) {
struct S S = { '\0', '\0','\0' };
S.Q.S[0] = 'A';
S.Q.S[1] = 'B';
printf("%d",strlen(S.Q.S));
return 0;
}
```

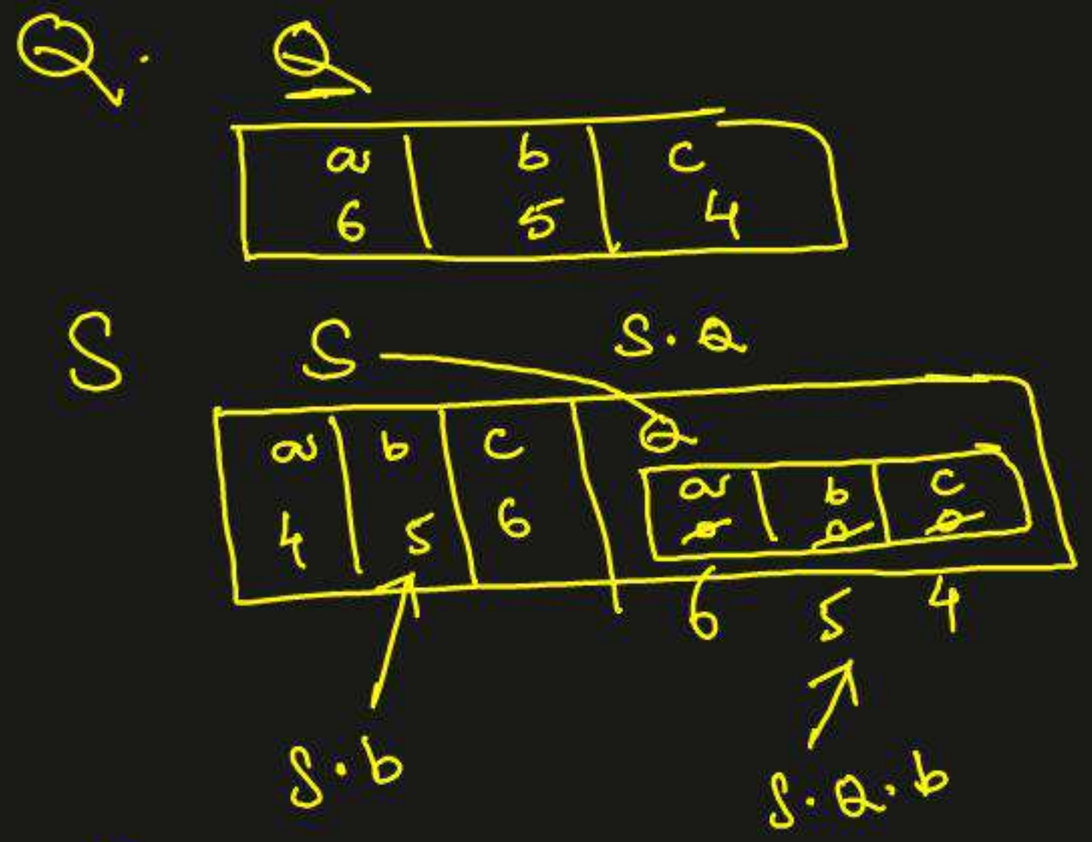


index of first NULL = 2

O/P: 2

Example - 3

```
#include <stdio.h>
struct Q {
int a,b,c;
};
struct S {
int a,b,c;
struct Q Q;
};
int main(void) {
struct Q Q = { 6,5,4 };
struct S S = { 4,5,6 };
S.Q = Q;
printf("%d",S.b - S.Q.b);
return 0;
}
```



$5 - 5 = 0$

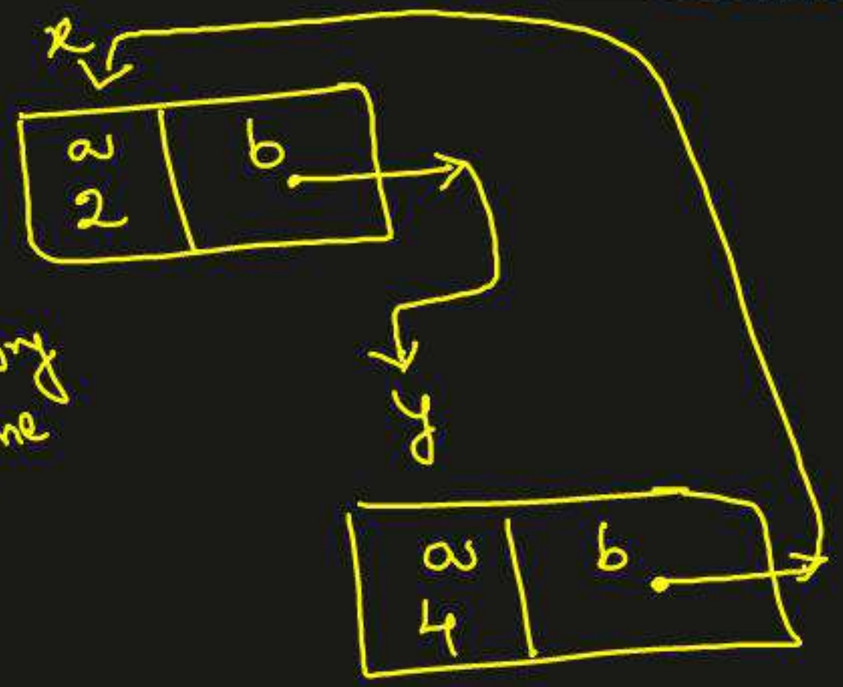
o/p: 0

Example - 4

```
#include <stdio.h>
#include <stdlib.h>
struct S {
int a;
struct S *b;
};
int main(void) {
struct S *x = (struct S*) malloc(sizeof(struct S));
struct S *y = (struct S*) malloc(sizeof(struct S));
x->a = 2;
x->b = y;
y->a = 4;
y->b = x;
printf("%d",x->b->b->b->b->a);
free(x); free(y);
return 0;
}
```

O/P : 2

Allocation of Memory at run time



$x \to b = y$

$y \to b = x$

$x \to b = y$

$y \to b = x$

$x \to a = 2$

Example - 5

```
#include<stdio.h>
int main()
{
struct num
{
int i, j, k, l;
};
struct num n = {1, 2, 3};
printf("%d %d %d %d", n.i, n.j, n.k, n.l);
}
```

n

i	j	k	l
1	2	3	0

1 2 3 0

A. 1 2 3 garbage value

B. 1 2 3 0

C. 1 2 3 3

D. None of the above

BYJU'S EXAM PREP

GATE 2000

Consider the following C declaration

```
struct {
short s[5]
union {
float y;
long z;
}u;
}t;
```

short s[5] → 2 Bytes each

10 Bytes Member 1

8 Bytes Member 2

float y; — 4 Bytes

long z; — 8 Bytes

}u; = 8 Bytes

Max{4, 8} = 8

Σ = 18 Bytes

t (18 Bytes)

Assume that the objects of the type short, float and long occupy 2 bytes, 4 bytes and 8 bytes, respectively. The memory requirement for variable t, ignoring alignment consideration, is

(A) 22 bytes

(B) 18 bytes

(C) 14 bytes

(D) 10 bytes

GATE 2000

The following C declarations

```
struct node
{
  int i;
  float j;
};
struct node *s[10] ;
define s to be
```

*s[10]

⇒ Array of 10 pointers, pointing to structure.

(A) An array, each element of which is a pointer to a structure of type node

(B) A structure of 2 fields, each field being a pointer to an array of 10 elements

(C) A structure of 3 fields: an integer, a float, and an array of 10 elements

(D) An array, each element of which is a structure of type node.

GATE 2018

Consider the following C program.

```
#include <stdio.h>
struct Ournode {
  char x, y, z;
};
int main() {
  struct Ournode p = {'1', '0', 'a' + 2};
  struct Ournode *q = &p;
  printf("%c, %c", *((char *)q + 1), *((char *)q + 2));
  return 0;
}
```

Char type

Structure type

A = 65
a = 97

x	y	z
1	0	c

q q+1 q+2

0 c

The output of this program is:

(A) 0, c

(B) 0, a+2

(C) '0', 'a+2'

(D) '0', 'c'

BYJU'S EXAM PREP

f(1) n=1
x=1 return 1.

Q1. Consider the following C function.

```
int fun ( int n ) {
  int x = 1, k ;
  if ( n == 1) return x ;
  for( k = 1 ; k < n ; ++ k )
    x = x + fun( k ) * fun( n - k ) ;
  return x ;
}
```

ANS: 51

The return value of fun (5) is _____.

f(1)=1 f(2)=2 f(3)=5
f(4)=15

fun(5) n=5
x=1

k=1 x=1+ f(1)*f(4) = 1+1×15 = 16

k=2 x= 16+ f(2)×f(3) = 16+ 2×5 =26

k=3 x= 26+ f(3)*f(2) = 26+5×2=36

k=4 x=36+ f(4)*f(1) = 36+15×1
= 51

return 51

[GATE 2015]

$f(4)$ $n=4$

$x = \cancel{1}\ \cancel{6}\ \cancel{10}\ 15$

$k=1$ $x = 1 + f(1) * f(3)$

$x = 1 + 1 * 5 = 1+5 = 6$

$k=2$ $x = 6 + f(2) * f(2)$

$= 6 + 2 \times 2 = 10$

$k=3$ $x = 10 + f(3) * f(1)$

$= 10 + 5 \times 1 = 15$

return x; $\Rightarrow$ return 15

$f(1) = 1.$

$f(2) = 2$

$f(3) = 5$

$f(4) = 15$

$f(3)$ $n=3$

$x = \cancel{1}\ \cancel{3}\ 5$

$k=1$ $x = 1 + f(1) * f(2)$

$x = 1 + 1 * 2 = 1 + 2 = 3$

$k=2$ $x = 3 + f(2) * f(1)$

$= 3 + 2 * 1 = 3 + 2 = 5$

return x; $\Rightarrow$ returns '5'

$f(2)$ $n=2$

$x=1$

$k=1$ $x = 1 + f(1) * f(1)$

$= 1 + 1 * 1 = 1 + 1 = 2$

$x=2$ returns '2'

BYJU'S EXAM PREP

Q2. Consider the C program below.

```
#include < stdio.h >
int *A, stkTop;
int stkFunc(int opcode, int val)
{
    static int size=0, stkTop=0;
    switch (opcode) {
        case -1: size = val; break;
        case 0: if (stkTop < size) A[stkTop++] = val; break;
        default: if (stkTop) return A[--stkTop];
    }
    return -1;
}
```

```
int main()
{
    int B[20]; A = B; stkTop = -1;
    stkFunc (-1, 10);
    stkFunc ( 0, 5);
    stkFunc ( 0, 10);
    printf ("%d\n", stkFunc(1, 0) + stkFunc(1, 0));
}
```

The value printed by the above program is _____.

[GATE 2015]

BYJU'S EXAM PREP

Home work

Q3. Consider the following C program

```
int tob(int b, int* arr) {
    int i;
    for (i=0; b>0; i++) {
        if (b%2) arr[i] = 1;
        else arr[i]=0;
        b = b/2;
    }
    return (i);
}
```

```
int pp(int a, int b) {
    int arr[20];
    int i, tot = 1, ex, len;
    ex = a;
    len = tob (b, arr);
    for (i=0;  i < len; i++) {
        if (arr[i] ==1)
            tot = tot * ex;
        ex = ex * ex;
    }
    return (tot);
}
```

The value returned by pp (3, 4) is _____.

[GATE 2020]

BYJU'S EXAM PREP

```
int tob(int b, int* arr) {
    int i;
    for (i=0; b>0; i++) {
        if (b%2) arr[i] = 1;
        else arr[i]=0;
        b = b/2;
    }
    return (i);
}
```

```
int pp(int a, int b) {
    int arr[20];
    int i, tot = 1, ex, len;
    ex = a;
    len = tob (b, arr);
    for (i=0;  i < len; i++) {
        if (arr[i] ==1)
            tot = tot * ex;
        ex = ex * ex;
    }
    return (tot);
}
```

BYJU'S EXAM PREP

Home-Work

Q4. Consider the following C program

```
int a, b, c = 0;
void prtFun (void);
int main ()
{
   static int a = 1; /* line 1 */
   prtFun();
   a += 1;
   prtFun();
   printf ( "\n %d %d " , a, b) ;
}

void prtFun (void)
{
   static int a = 2; /* line 2 */
   int b = 1;
   a += ++b;
   printf (" \n %d %d " , a, b);
}
```

What output will be generated by the given code segment ?

[GATE 2012]

A
3 1
4 1
4 2

B
4 2
6 1
6 1

C
4 2
6 2
2 0

D
3 1
5 2
5 2

BYJU'S EXAM PREP

Home-work

Q5. Consider the following C code segment.

```
int a, b, c = 0;
void prtFun(void);
main()
{
   static int a = 1;      /* Line 1 */
   prtFun();
   a += 1;
   prtFun();
   printf(" \n %d %d ", a, b);
}
void prtFun(void)
{
   static int a = 2;      /* Line 2 */
   int b = 1;
   a += ++b;
   printf(" \n %d %d ", a, b);
}
```

What output will be generated by the given code segment if:

Line 1 is replaced by auto int a=1;

Line 2 is replaced by register int a=2;

[GATE 2012]

A
3 1
4 1
4 2

B
4 2
6 1
6 1

C
4 2
6 2
2 0

D
4 2
4 2
2 0

BYJU'S EXAM PREP

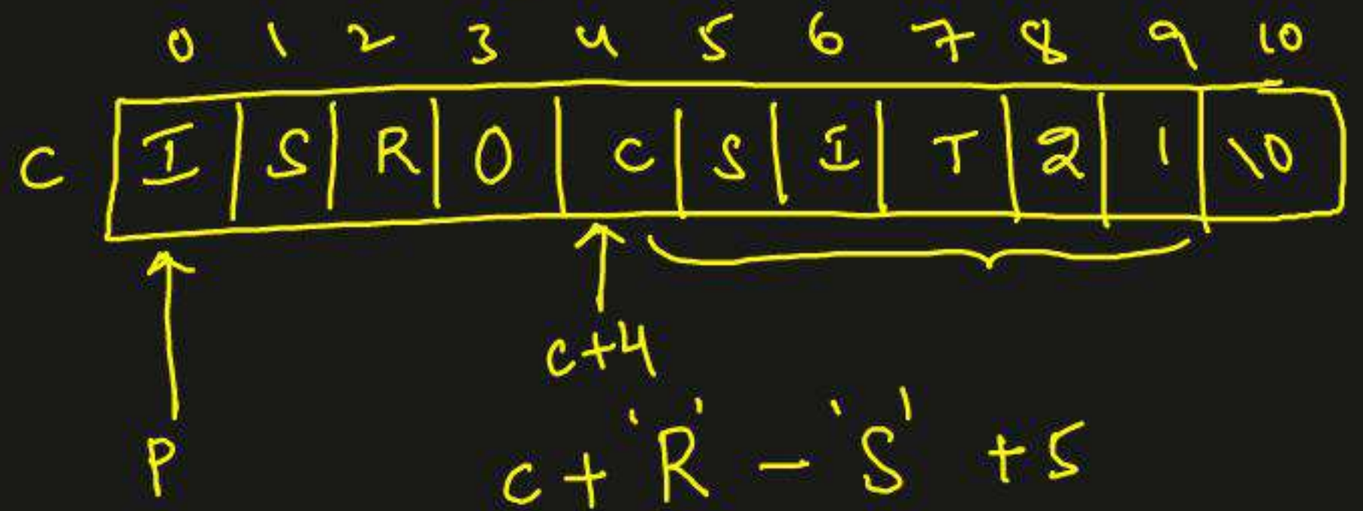
$p[2] == 2[p]$

$p[i] == i[p] == c[i] == i[c]$

Q6. Consider the following C function :

```
#include <stdio.h>
int main(void)
{
char c[ ] = "ISROCSIT21";
char *p=c;
printf("%s", c+2[p] – p[1] + 5);
return 0;
}
```

The output of the program is _____.



$c + 'R' - 'S' + 5$

$c + (-1) + 5 = c + 4$

A. SIT21

B. OCSIT21

C. CSIT21

D. ISROC

BYJU'S EXAM PREP

Q7. Consider the following C program:

```
#include <stdio.h>
int main( )
{
int a[4][5] =          { {1, 2, 3, 4, 5},
                         {6, 7, 8, 9, 10},
                         {11, 12, 13, 14, 15},
                         {16, 17, 18, 19, 20} };

printf("%d"\n,*(*(a+**a+1)+4));
return(0);
}
```

a	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

**a ⇒ *(*(a+0)+0)
= a[0][0] = 1

a+1+1 = a+2
((a+2)+4) == a[2][4]

A. 10

B. 14

C. 15

D. 19

i[j] = i+j

BYJU'S EXAM PREP

Q8. What will be the output of the C program, if executed on a 32-bit processor ?

```
#include<stdio.h>
void function(char**);
int main()
{
    char *arr[] = { "ant", "bat", "cat", "dog", "egg", "fly" };
    function(arr);
    return 0;
}

void function(char **ptr)
{
    char *ptr1;
    ptr1 = (ptr += sizeof(int))[-2];
    printf("%s\n", ptr1);
}
```

ptr1 = (ptr = ptr + sizeof(int)[-2])
= ptr + 4[-2]
= ptr + (4-2) = ptr + 2

0 1 2 3 4 5

ptr1

A bat

B cat

C egg

D dog

BYJU'S EXAM PREP

Home-work

Q9. Consider the following C program :

```
#include < stdio.h >
 int main () {
        int arr [] = {1,2,3,4,5,6,7,8,9,0,1,2,5}, *ip = arr+4;
         ip= ip++;
         printf ("%d\n", ip[1]);
         return 0;
}
```

The number that will be displayed on execution of the program is _____.

A. 5

B. 6

C. 7

D. 8

ANSWERS:

3.

4.

5.

9.

BYJU'S EXAM PREP

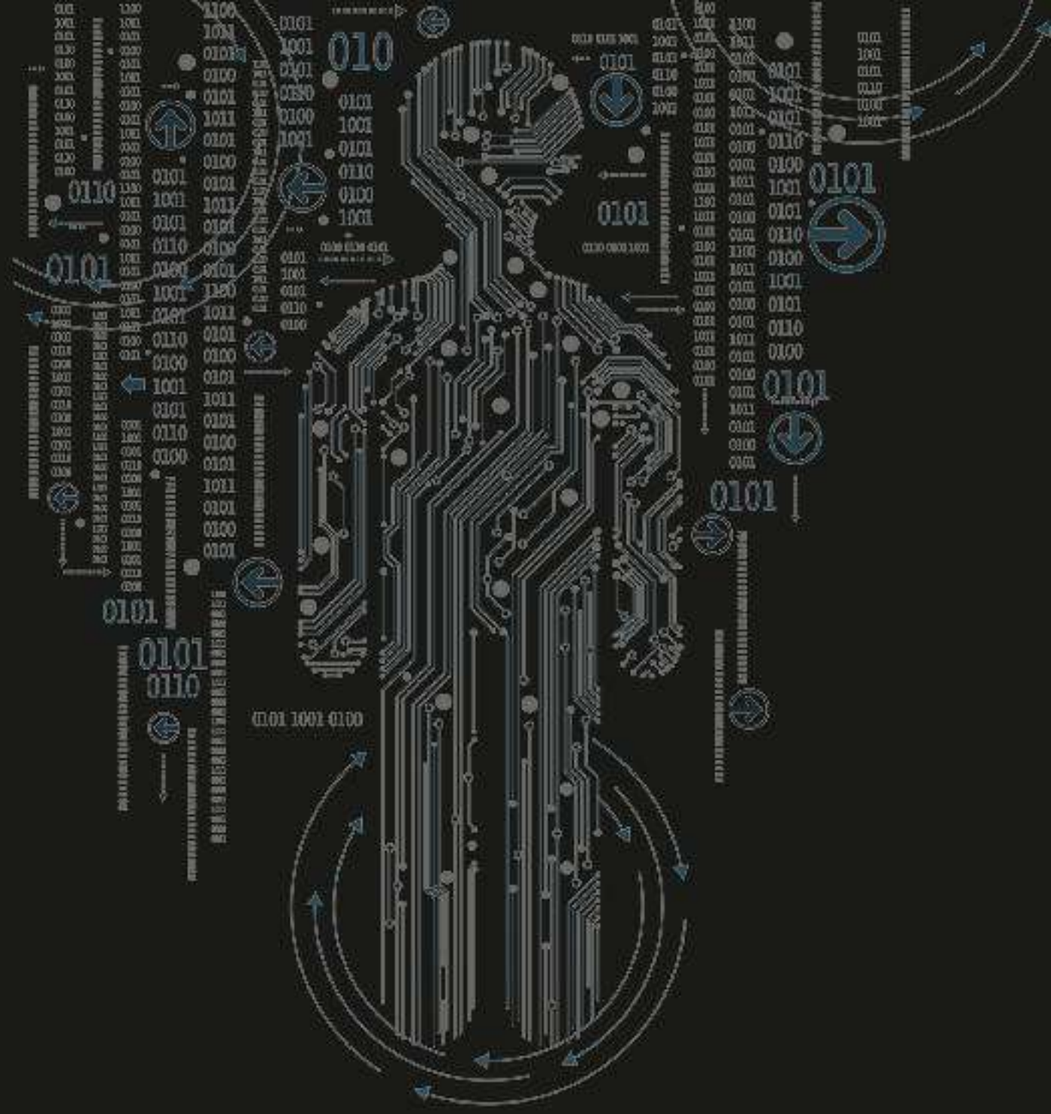
4.5 Google Play Review

GET IT ON Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Interactive Session -1



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

ANS: 81

Q3. Consider the following C program

```
int tob(int b, int* arr) {
    int i;
    for (i=0; b>0; i++) {
        if (b%2) arr[i] = 1;
        else arr[i]=0;
        b = b/2;
    }
    return (i);
}
```

```
int pp(int a, int b) {
    int arr[20];
    int i, tot = 1, ex, len;
    ex = a;
    len = tob (b, arr);
    for (i=0;  i < len; i++) {
        if (arr[i] ==1)
            tot = tot * ex;
        ex = ex * ex;
    }
    return (tot);
}
```

The value returned by pp (3, 4) is _____.

[GATE 2020]

3 4

0 1 2

arr: 0 0 1 -- --

ex = 3

len = 3

o/p = 3

i = 2 1 > 0 True

1 % 2 == 1 True

arr[2] = 1

b = 1/2 = 0

i = 3 b > 0 False

i = 0 b > 0

b % 2 == 0 False arr[0] = 0

b = 4/2 = 2

i = 1 2 > 0 2 % 2 == 0 False

arr[1] = 0 b = 2/2 = 1

ANS: 81

```
int tob(int b, int* arr) {
    int i;
    for (i=0; b>0; i++) {
        if (b%2) arr[i] = 1;
        else arr[i]=0;
        b = b/2;
    }
    return (i);
}
```

```
int pp(int a, int b) {
    int arr[20];
    int i, tot = 1, ex, len;
    ex = a;
    len = tob (b, arr);
    for (i=0;  i < len; i++) {
        if (arr[i] ==1)
            tot = tot * ex;
        ex = ex * ex;
    }
    return (tot);
}
```

ex 3 9 81 6561

tot 1 81

ex = 3

len = 3

i = 0 0 < 3

if (arr[0] == 1) False

i = 1 1 < 3

if (arr[1] == 1) False ex = ex * ex = 9 * 9 = 81

i = 2 2 < 3 True

if (arr[2] == 1) True

tot = 1 * 81 = 81

ex = ex * ex = 81 * 81

i = 3

3 < 3

False.

returns tot = 81

Global variables

a: 0 b: 0 c: 0

NOTE: Static variables are initialized only once.

Q4. Consider the following C program

```
int a, b, c = 0;
void prtFun (void);
int main ()
{
   static int a = 1; /* line 1 */
   prtFun();
   a += 1;
   prtFun();
   printf ( "\n %d %d " , a, b) ;
}

void prtFun (void)
{
   static int a = 2; /* line 2 */
   int b = 1;
   a += ++b;
   printf (" \n %d %d " , a, b);
}
```

static a ~~1~~ 2

a = a+1

static a ~~2~~ 4 6

a = a + ++b

2 + 2 = 4

4, 2

b ~~1~~ 2

b 2

a = 4 + 2 = 6

o/p:

4 2

6 2

2 0

What output will be generated by the given code segment ?

[GATE 2012]

(A) 3 1
4 1
4 2

(B) 4 2
6 1
6 1

(C) 4 2
6 2
2 0

(D) 3 1
5 2
5 2

Q5. Consider the following C code segment.

```
int a, b, c = 0;
void prtFun(void);
main()
{
  static int a = 1;      /* Line 1 */
  prtFun();
  a += 1;
  prtFun();
  printf(" \n %d %d ", a, b);
}
void prtFun(void)
{
  static int a = 2;      /* Line 2 */
  int b = 1;
  a += ++b;
  printf(" \n %d %d ", a, b);
}
```

What output will be generated by the given code segment if:

Line 1 is replaced by auto int a=1;

Line 2 is replaced by register int a=2;

[GATE 2012]

(A)
3 1
4 1
4 2

(B)
4 2
6 1
6 1

(C)
4 2
6 2
2 0

(D)
4 2
4 2
2 0

Q6. Consider the following C function :

```
#include <stdio.h>
int main(void)
{
char c[ ] = "ISROCSIT21";
char *p=c;
printf("%s", c+2[p] – p[1] + 5);
return 0;
}
```

The output of the program is _____.

A. SIT21

B. OCSIT21

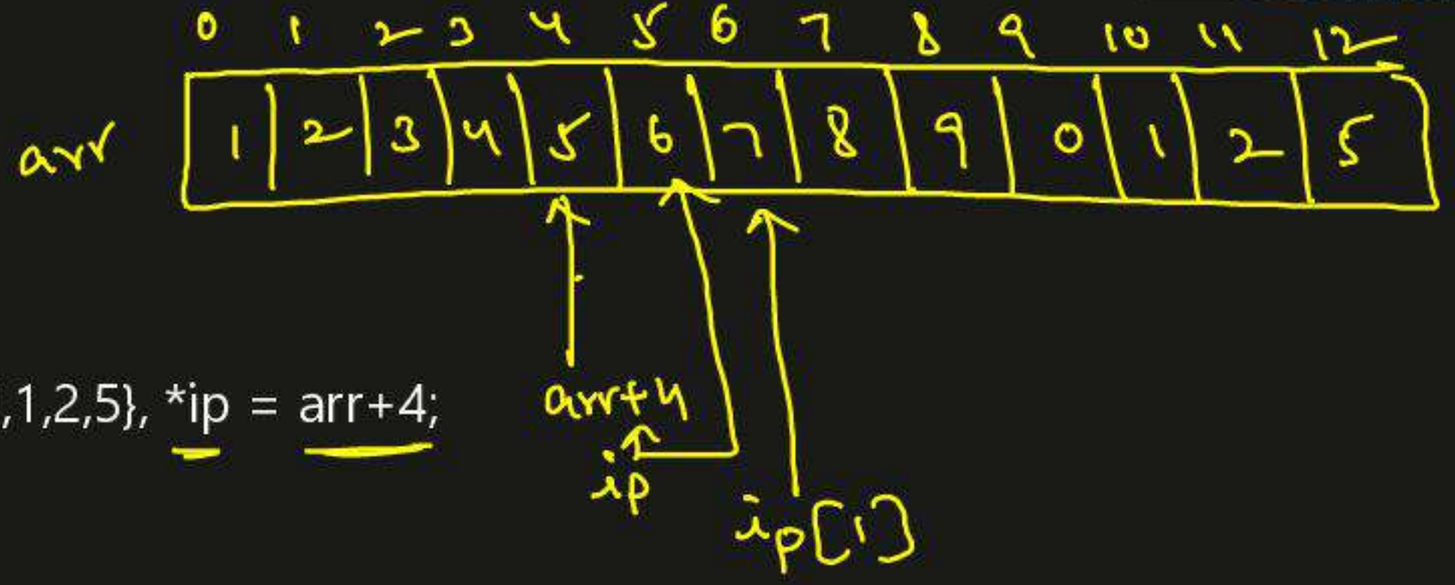
C. CSIT21

D. ISROC

a = b++;
a = a++;

Q9. Consider the following C program :

```
#include < stdio.h >
 int main () {
        int arr [] = {1,2,3,4,5,6,7,8,9,0,1,2,5}, *ip = arr+4;
         ip= ip++;
         printf ("%d\n", ip[1]);
         return 0;
}
```



The number that will be displayed on execution of the program is _____.

A. 5

B. 6

C. 7

D. 8

BYJU'S EXAM PREP

Concepts Covered So Far

- 'C' Tokens
- Data Types
- I/O Statements
- Control Statements
- Pointers
- Arrays
- Functions
- Structures
- Unions

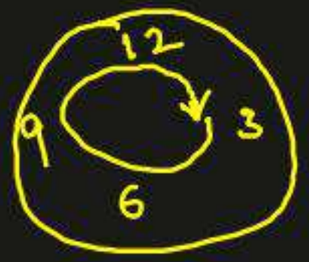
Interactive Session - 1

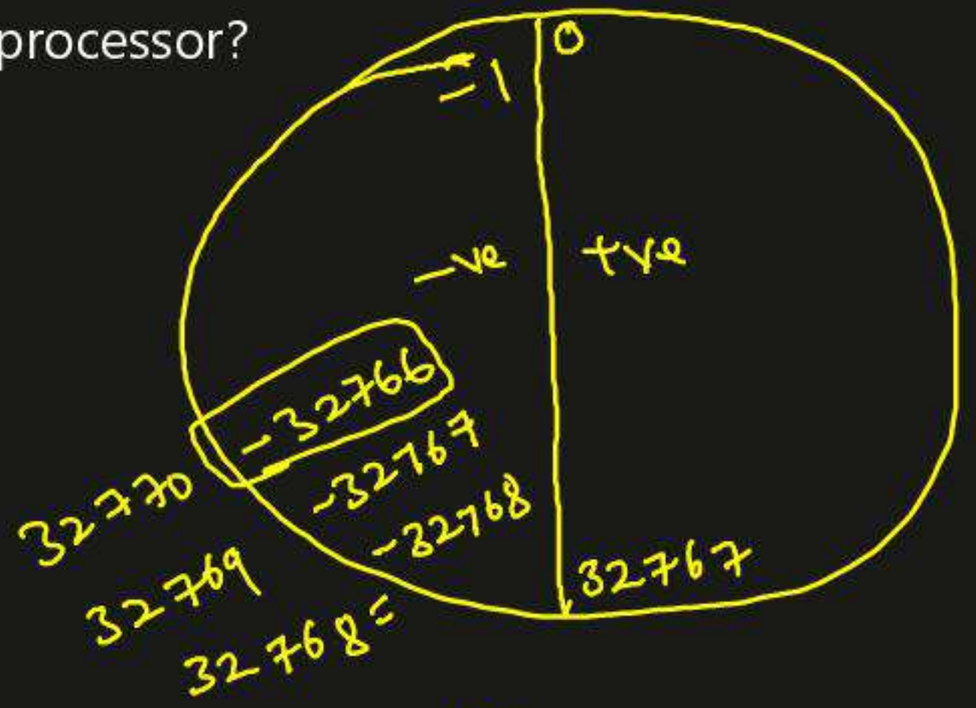
Signed int (16 bits) = -2^{15} to $(2^{15}-1)$
-32768 to 32767

Q1. What is output of below code, if run on a 16-bit processor?

```
void main()
{
int a=32770;
printf("%d",a);
}
```

signed



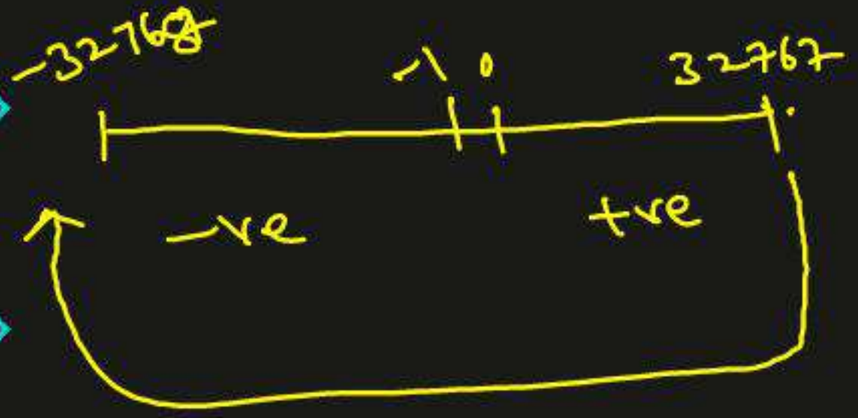


A. 3

B. -3

C. -32766

D. 32766



BYJU'S EXAM PREP

Q2. Consider the declarations below, to use on a 16-bit processor:

signed int a; -32768 to 32767

unsigned int b; 0 to 65535 Max.

signed int c; -32768 to 32767

The Maximum Values that are processed as it is for a,b,c are respectively __ , __ , __.

A. 32767, 65535, 32768

B. 65535, 32767, 32767

C. 32767, 65535, 32767

D. 32767, 32767, 65535

BYJU'S EXAM PREP

Q3. What is output of below program on input "ISRO 2021 EXAM"

```
void main()
{
char s[10];;
scanf("%s",s);
printf("%s",s);

}
```

scanf() Terminates here.

ISRO _ 2021 _ EXAM

: ISRO

	0	1	2	3	4	5	6	7	8	9
S	I	S	R	O	\0	\0	\0	\0	\0	\0

strlen(s) = 4 (Index of first NULL)

sizeof(s) = 10 (Memory allocated)

A. ISRO

B. ISRO 2021 E

C. ISRO 2021

D. ISRO 2021 EXAM

Interactive Session - 1

BYJU'S EXAM PREP

Q4. What is a's value?

```
void main()
{
int a;
a=printf("ISRO\nBARC\tNIELIT");
}
```

Single character:
- \0 = NULL character
- \n = New line character
- \t = Tab space character (4 spaces)

} Escape Sequences (special Pre defined characters)

a = scanf("%d", &a);

a = 1

4235

16 characters processed.

O/P: ISRO

BARC ---- NIELIT

(A) 14

(B) 15

(C) 16

(D) 18

Printf(): returns No. of characters Processed

scanf(): No. of inputs Processed

NOTE: Printf(), scanf() return value is always unsigned Integer.

```
int a, b, i;
char x[20];
float c;
a = scanf("%d %s %f", &b, &x, &c);
i = printf("%d %s %f", b, x, c);
```

Let b = 1857
x = "BYJUSEXAMPREP"
c = 0.123456

a = 3

i = 25

o/p: 1857BYJUSEXAMPREP0.123456

BYJU'S EXAM PREP

Q5. What is a's value?

```
void main()
{
char Char='H';
int a=printf("%d",Char);
}
```

H

'H' Printing as Decimal integer

⇒ H ASCII value is Printed

⇒ o/p: 72

a value = 2

A 72

B 722

C 2

D 272

BYJU'S EXAM PREP

Q6. What is the output of below program?

```
void main()
{
int a=0X78;
printf("%o",a);
}
```

Ex:
653 : Decimal
0653 : octal
0X653 : Hexa

Prefix 0 : octal Integer

Prefix 0X (or) 0x : Hexa Decimal Integer

Hexa → octal

Hexa to Binary
7 8
0111 1000
⇒ 1 7 0

octal Integer

%d (or) %i Decimal integer

%o octal

%x Hexa Decimal.

A 64

B 78

C 110

D 170

BYJU'S EXAM PREP

Q7. What is the output of below code?

```
void main()
{
if(printf(""))
printf("hai");
else if(printf(""))
printf("Bye");
}
```

No character is Processed, so, it returns zero.

⇒ if (0) False

→ if (0) ⇒ False

not executed.

A. hai

B. Bye

C. No Output

D. Error

BYJU'S EXAM PREP

Home work

Q8. What will be the output of below program?

```
#include <stdio.h>
int main(void) {
int a=0,b=1;
if(a,b)
printf(" GOOD ");
else
printf( " BAD ");
return 0;
}
```

A. GOOD

B. BAD

C. No Output

D. Compiler Error

Interactive Session - 1

Home work

Q9. Which of the following is/are the valid syntax of for loop?

1. for(i;j;k)
2. for(i;j;)
3. for(i; ;k)
4. for(; ;k)

A. 1 only

B. 1,2,3

C. 1, 2, 3,4

D. All are Invalid

BYJU'S EXAM PREP

Q10.

```
int main()
{
    int a;
    for(a=printf("%d", a);a>=3;a++)
    {
            //
    }
    return 0;
}
```

The number of times the above loop executes for a=456?

A. 0

B. 1

C. 3

D. Error

BYJU'S EXAM PREP

Q11. What is the output of below code?

```
void main()
{
int i,j,sum=0;
for(i=0;i<5;i++);
for(j=5;j>=0;j/=2)
sum++;
printf("%d",i+j-sum);
}
```

A. 2

B. 4

C. 7

D. 8

BYJU'S EXAM PREP

t/w

Q12. What does the following program print?

```
void f(int *p, int *q)
{
  p = q;
 *p = 2;
}
int i = 0, j = 1;
int main()
{
  f(&i, &j);
  printf("%d %d \n", i, j);
  getchar();
  return 0;
}
```

A. 2 2

B. 2 1

C. 0 1

D. 0 2

BYJU'S EXAM PREP

Q13.
```
void main()
{
int x[ ]={11,22,33,44,55,66,77,88,99},*y;
y=&x;
y=y+1;
*y=y[1];
y[5]=y[1];
y[6]=y[3] - y[2];
y[7]=y[4] / 3;
printf("%d",x[7]);
}
```

What will be the output?

4/w

A. 77

B. 88

C. 11

D. 22

H/w

Q14. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        struct node
        {
        int a, b, c;
        };
        struct node num = {3, 5, 6};
        struct node *ptr = & num;
        printf("%d\n", *((int*)ptr + 1 + (3-2)));
        return 0;
}
```

A. 3

B. 5

C. Compilation error

D. 6

BYJU'S EXAM PREP

H/w

Q15. What will be the output of the C program?

```
#include<stdio.h>
#include<string.h>
struct player
{
    char pname[20];
}pl;
char* play(struct player *temp_pl)
{
    strcpy(temp_pl->pname, "virat");
    return temp_pl->pname;
}
int main()
{
    strcpy(pl.pname, "rohit");
    printf("%s %s", pl.pname, play(&pl));
    return 0;     }
```

A. rohit virat

B. rohit rohit

C. virat rohit

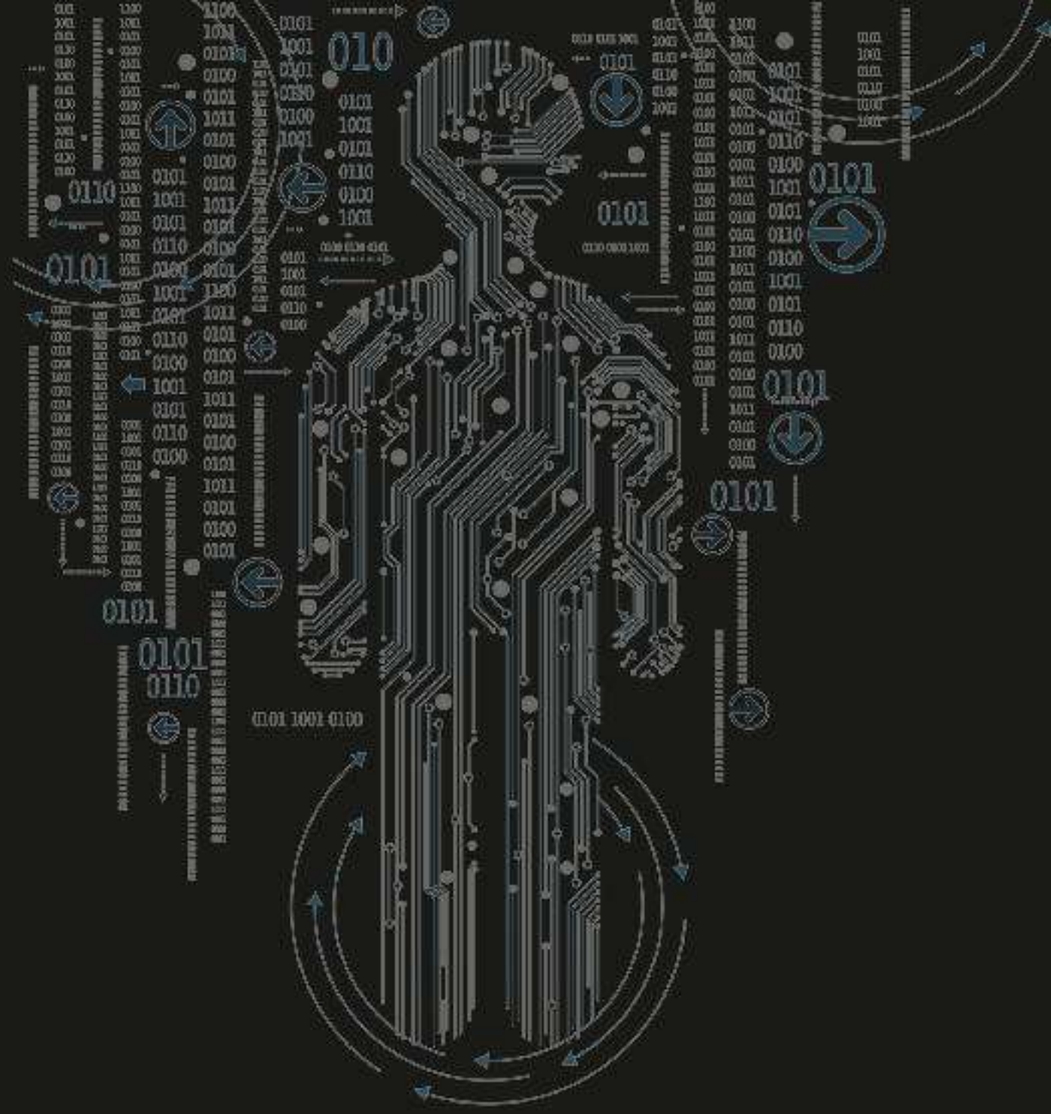
D. virat virat

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Implementation of Stack



Guidelines to Attend Live Class

BYJU'S EXAM PREP

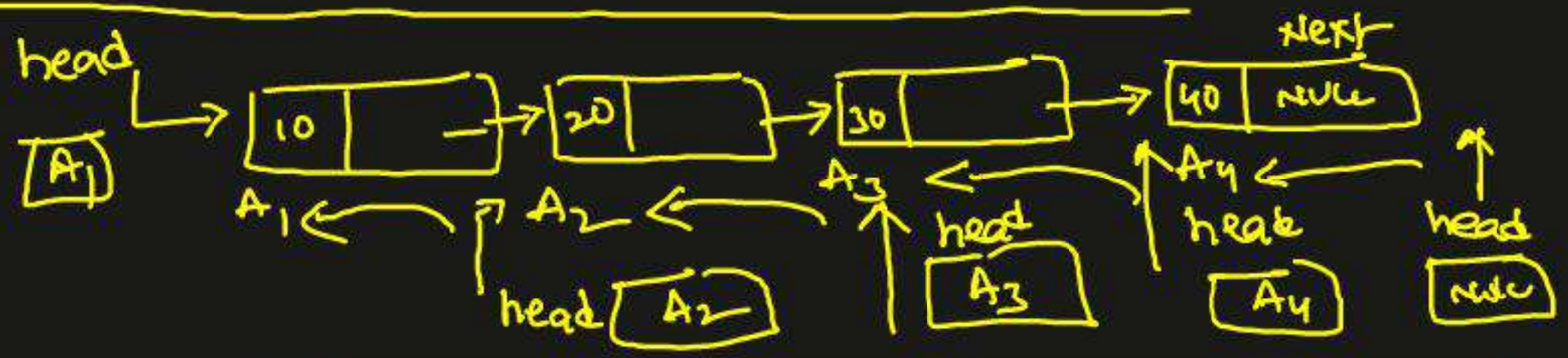
- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Implementation Of Stack

Q1. What does the following function do for a given Linked List with first node as head ?

```
void fun1(struct node* head)
{
    if(head == NULL)
    return;
    fun1(head->next);
    printf("%d  ", head->data);
}
```

o/p: 40, 30, 20, 10



A. Prints all nodes of linked lists

B. Prints all nodes of linked list in reverse order

C. Prints alternate nodes of Linked List

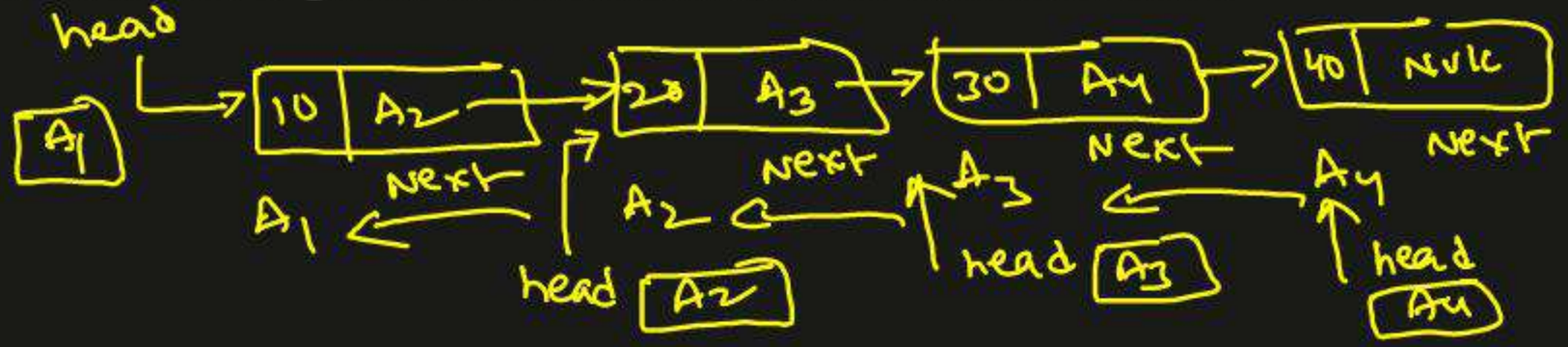
D. Prints alternate nodes in reverse order

Implementation Of Stack

Q2. What does the following function do for a given Linked List with first node as head?

```
void fun1(struct node* head)
{
    if(head->next == NULL)
    return;
    printf("%d  ", head->next->data);
    fun1(head->next);
}
```

o/p. 20, 30, 40,



A. Prints all nodes of linked lists

B. Prints all nodes of linked list except first node

C. Prints alternate nodes of Linked List

D. Prints all nodes of linked list except last node

Implementation Of Stack

No. of Comparisions
Best — 1
Avg — n/2
Worst — n

Q3. In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is -

[GATE 2002]

A. log n

B. n/2

C. logn – 1

D. n

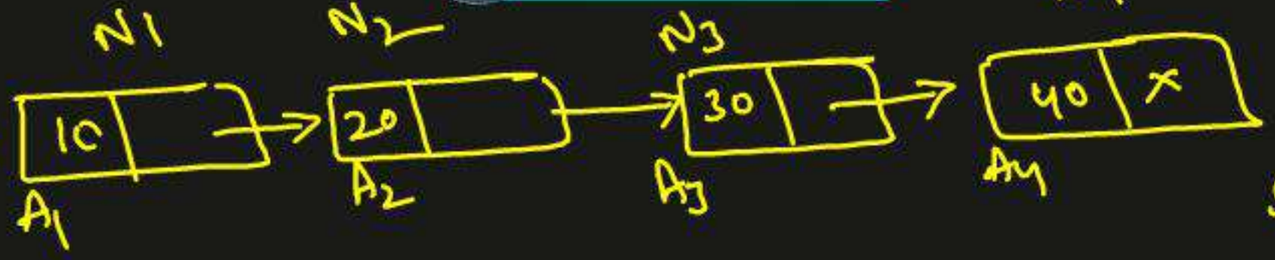
10 == 40
20 == 40
30 == 40
40 == 40
Search Success

10 == 70
20 == 70
30 == 70
40 == 70
Search unsuccessful

Best-case : O(1)

Avg. Case
O(n/2)
== O(n)
Complexity.

Worst-case : O(n)



Key = 40 (or) key = 70

if (temp ~~head~~ -> data == key)
Search success

else temp = temp ~~head~~ = ~~head~~ -> next;

Implementation Of Stack

BYJU'S EXAM PREP

Q4. The following C function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

[GATE 2008]

```
void rearrange(struct node *list)  {
  struct node *p, * q;
  int temp;
  if ((!list) || !list->next)        return;
  p = list;     q = list->next;
  while(q)     {
     temp = p->value;  p->value = q->value;
     q->value = temp;   p = q->next;
     q = p?p->next:0;     }  }
```

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

A. 1,2,3,4,5,6,7

B. 2,1,4,3,6,5,7

C. 1,3,2,5,4,7,6

D. 2,3,4,5,6,7,1

F || F == False

if(!list || !list->next)

return;

p=list;

q=list->next;

while(q) < NULL False / Non-NULL True

{ temp= p->value;

p->value = q->value

q->value = temp

p= q->next;

q = [p ? p->next : 0]

True: q = p->next

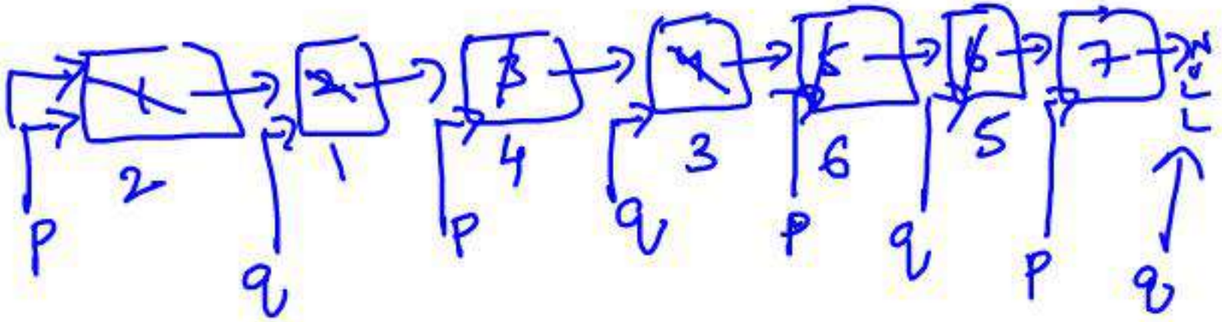
False: q=0

}

!list => if NO elements

!list->next => if one element

List



temp

list: 2, 1, 4, 3, 6, 5, 7

Implementation Of Stack

$O(n-2) == O(n)$

Q5. Let P be a singly linked list. Let Q be the pointer to an intermediate node x in the list. What is the worst-case time complexity of the best-known algorithm to delete the node x from the list ?

[GATE 2004 IT]

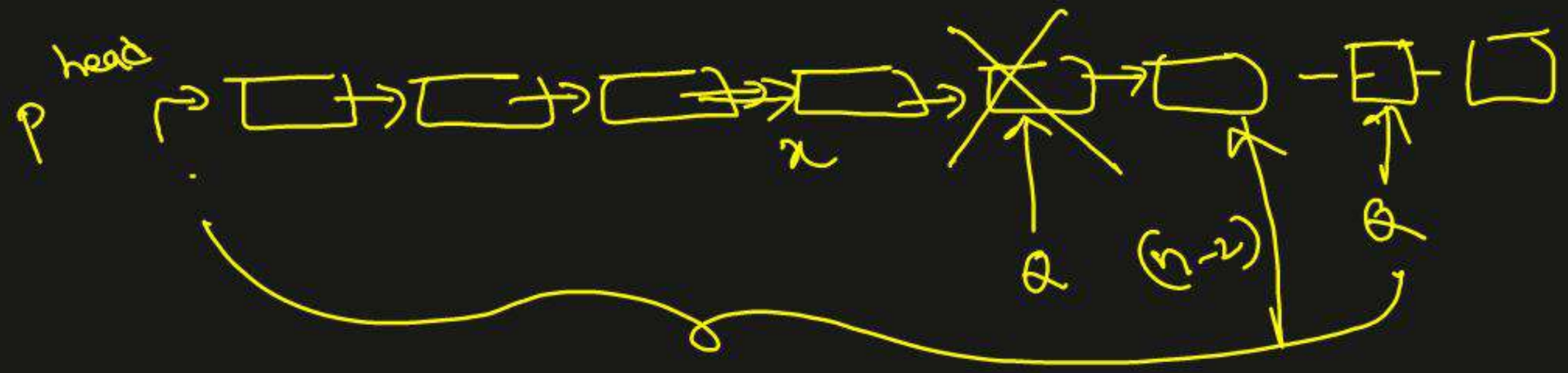
A. O(n)

B. $O(\log^2 n)$

C. O(logn)

D. O(1)

Last but one Node.



Implementation Of Stack

Home-work

1, 2, 3, 4, 5

o/p:- 5, 1, 2, 3, 4

Q6. The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. [GATE 2010]

```
typedef struct node  {
  int value;
  struct node *next;
}Node;
Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
    return head;
  q = NULL; p = head;
  while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  ____________________________
  return head;
}
```

Implementation Of Stack

Choose the correct alternative to replace the blank line.

A. q = NULL; p->next = head; head = p;

B. q->next = NULL; head = p; p->next = head;

C. head = p; p->next = q; q->next = NULL;

D. q->next = NULL; p->next = head; head = p;

Implementation Of Stack

A. q = NULL; p->next = head; head = p;

B. q->next = NULL; head = p; p->next = head;

C. head = p; p->next = q; q->next = NULL;

D. q->next = NULL; p->next = head; head = p;

```
Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
    return head;
  q = NULL;
p = head;
while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  ____________________________
  return head;
}
```

NOTE:

WEDNESDAY 20/April/2022 : No classes

will be rescheduled

@ 11AM
- 23/April/2022 : one class
- 24/April/2022 : one class

Implementation Of Stack

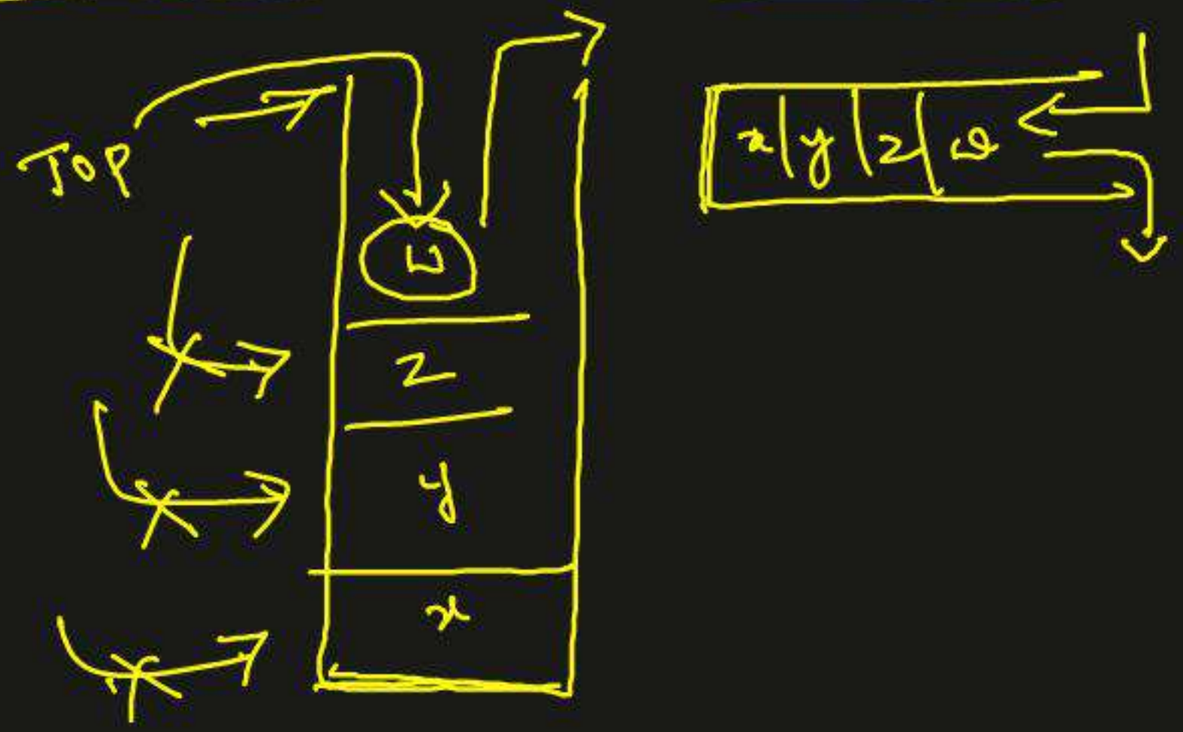
STACK

A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle.

A stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

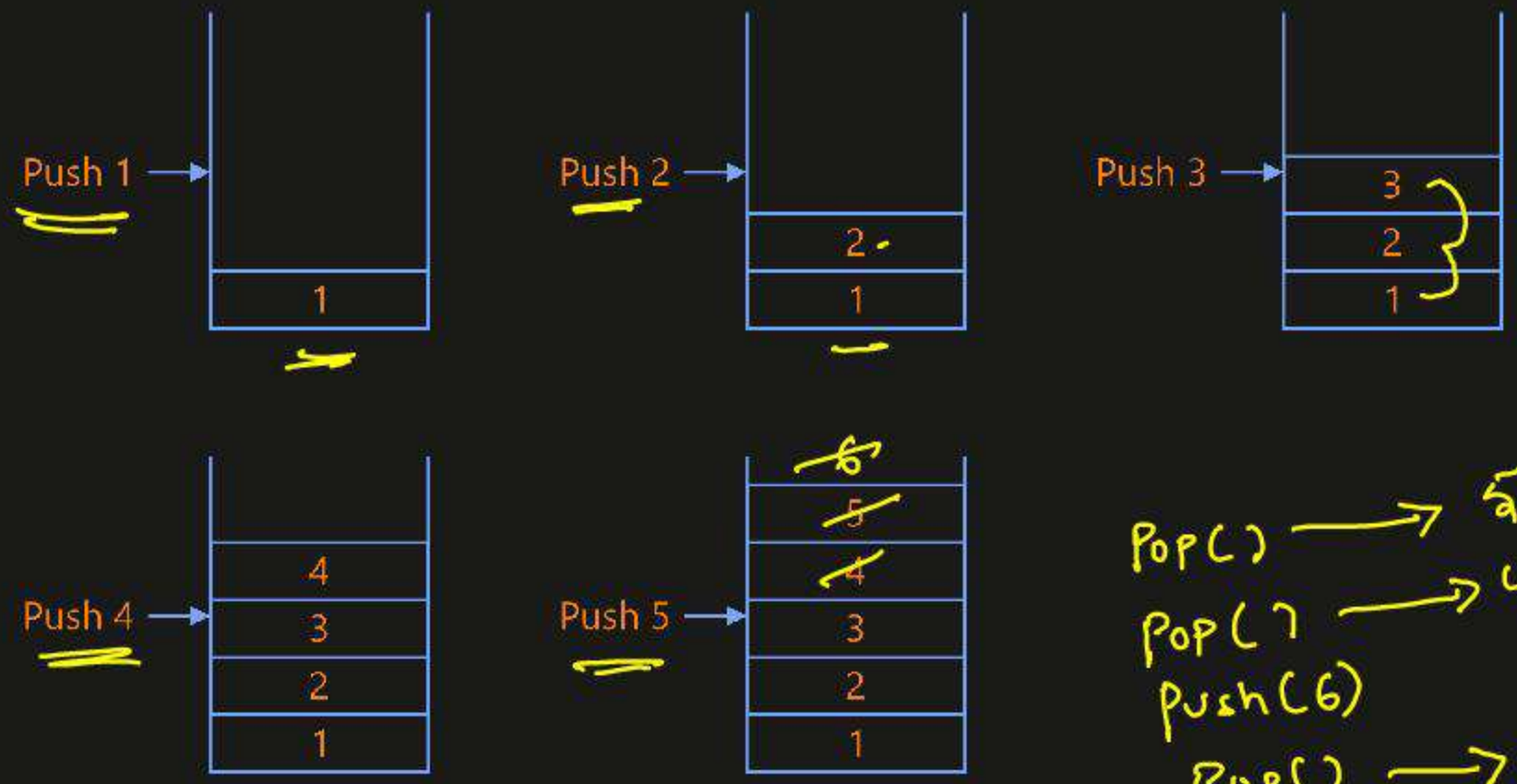
Stack Operations:

- Insertion Operation : Push Operation
- Deletion Operation: Pop Operation
- Access Operation: Peek Operation



Implementation Of Stack

STACK



Stack

Pop() → 5
Pop() → 4
Push(6)
Pop() → 6
Peek() → Printed 3

Implementation Of Stack

Operations On Stack

The following are some common operations implemented on the stack:

- push() : When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- pop() : When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- isEmpty() : It determines whether the stack is empty or not.
- isFull() : It determines whether the stack is full or not.'
- peek() : It returns the element at the given position.
- count() : It returns the total number of elements available in a stack.
- change() : It changes the element at the given position.
- display() : It prints all the elements available in the stack.

Implementation Of Stack

x = Peek();

display → Printf("%d", stack[top]);

Possible Sequences

→ Assume Elements A, B, C, D, E are Pushed in the same sequence. But Pop operation can be done at any time. What are different Possible sequences of o/p?

① E, D, C, B, A — Push(A), Push(B), Push C, Push D, Push E, : valid
Pop, Pop, Pop, Pop, Pop

② A, B, C, D, E — Push(A), Pop, Push(B), Pop, Push C, Pop, Push D, Pop, Push E, Pop — valid

③ C, B, A, E, D — Push A, Push B, Push C, Pop, Pop, Pop, Push D, Push E, Pop, Pop — valid

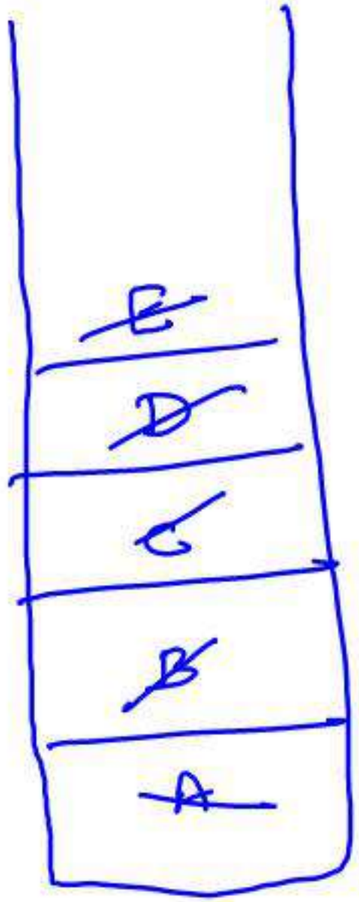
(4) E, B, A, C, D

Invalid.

Push A, Push B, Push C, Push D,
Push E, Pop : E
Pop : D but not B.

(5) A, B, D, C, E

Push A, Pop, Push B, Pop,
Push C, Push D, Pop, Pop
Push E, Pop



Push A
Push B
Push C → Pop – C
Push D → Pop() D

(6) D, C, A, B, E
Invalid.

o/p: A, B, D, C, E Valid.

BYJU'S EXAM PREP

Implementation Of Stack

A Stack can be implemented using either of the following data structures:

- An Array
- Linked List
- Queue

Implementation Of Stack

Stack using Array - Push

```
void push (int val,int n) //n is size of the stack
{
    if (top == n -1)
    printf("\n Overflow");
    else
    {
    top = top +1;
    stack[top] = val;
    }
```

top

-X 0 1

Let size, n = 5

Initially, stack empty: top = -1

val n

Push(30, 5)

if(-1 == 4)

False

else

top = 0

stack[0] = 30;

val n

Push(40, 5)

0 == 4 False

else

top = 1

stack[1] = 40

0	1	2	3	4
30	40	50	60	70.

Stack:

val n

Push(80,5)

if(4 == 4) True

	top
70	4
60	3
50	2
40	1
30	0

top = -1

Implementation Of Stack

top = ~~4~~ ~~3~~ 2

Stack using Array - Pop

Now, Let stack contains 5 elements (Pushed 5 elements).

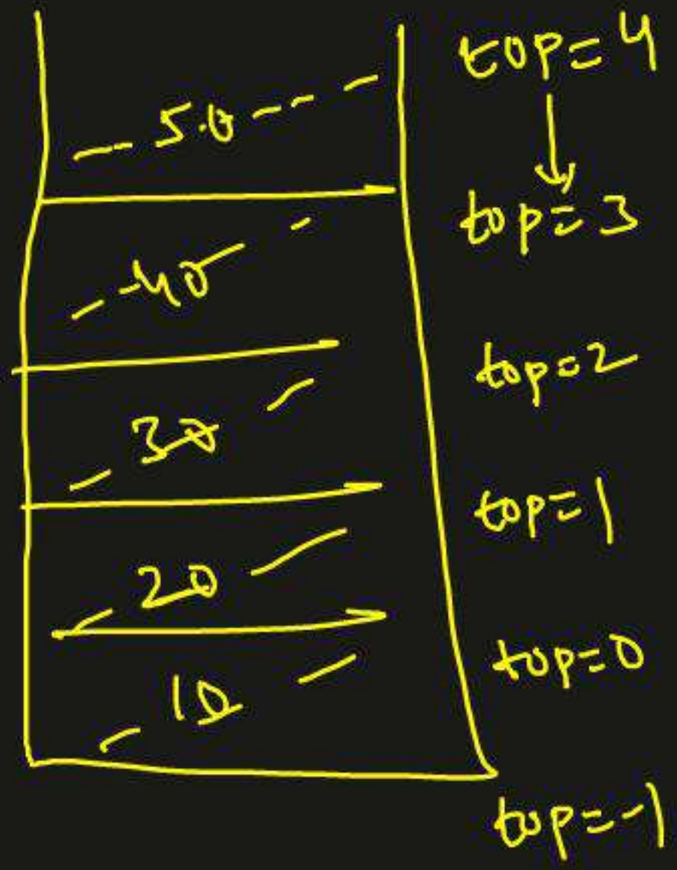
```
int pop ()
{
  if(top == -1)
  {
    printf("Underflow");
    return 0;
  }
  else
  {
    return stack[top - - ];
  }
}
```

Pop( )
(4 == -1) F
return stack[top]
return 50

Pop( ) (3 == -1) F
return stack[3] = 40

∴ Pop(), Pop(), Pop()
30 20 10
top = 1 top = 0 top = -1

Now, Pop() : stack is empty.



BYJU'S EXAM PREP

Stack using Array - Peek

```
int peek()
{
    if (top == -1)
    {
        printf("Underflow");
        return 0;
    }
    else
    {
        return stack [top];
    }
}
```

Implementation Of Stack

Stack using LL - Push

Linked List use Pointers.
Pointers get memory at runtime.
Hence, when required memory keep allocating.
So, LL Implementation, NO overflow condition.
Assuming there is sufficient memory space,

To be contd...

```
void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("not able to push the element");
    }
    else
    {
        printf("Enter the value");
        scanf("%d",&val);
        if(head==NULL)
        {
            ptr->val = val;
            ptr -> next = NULL;
            head=ptr;
        }
```

Implementation Of Stack

Stack using LL - Pop

```
void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->val;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped");

    }
}
```

BYJU'S EXAM PREP

Stack using LL – Peek / Display

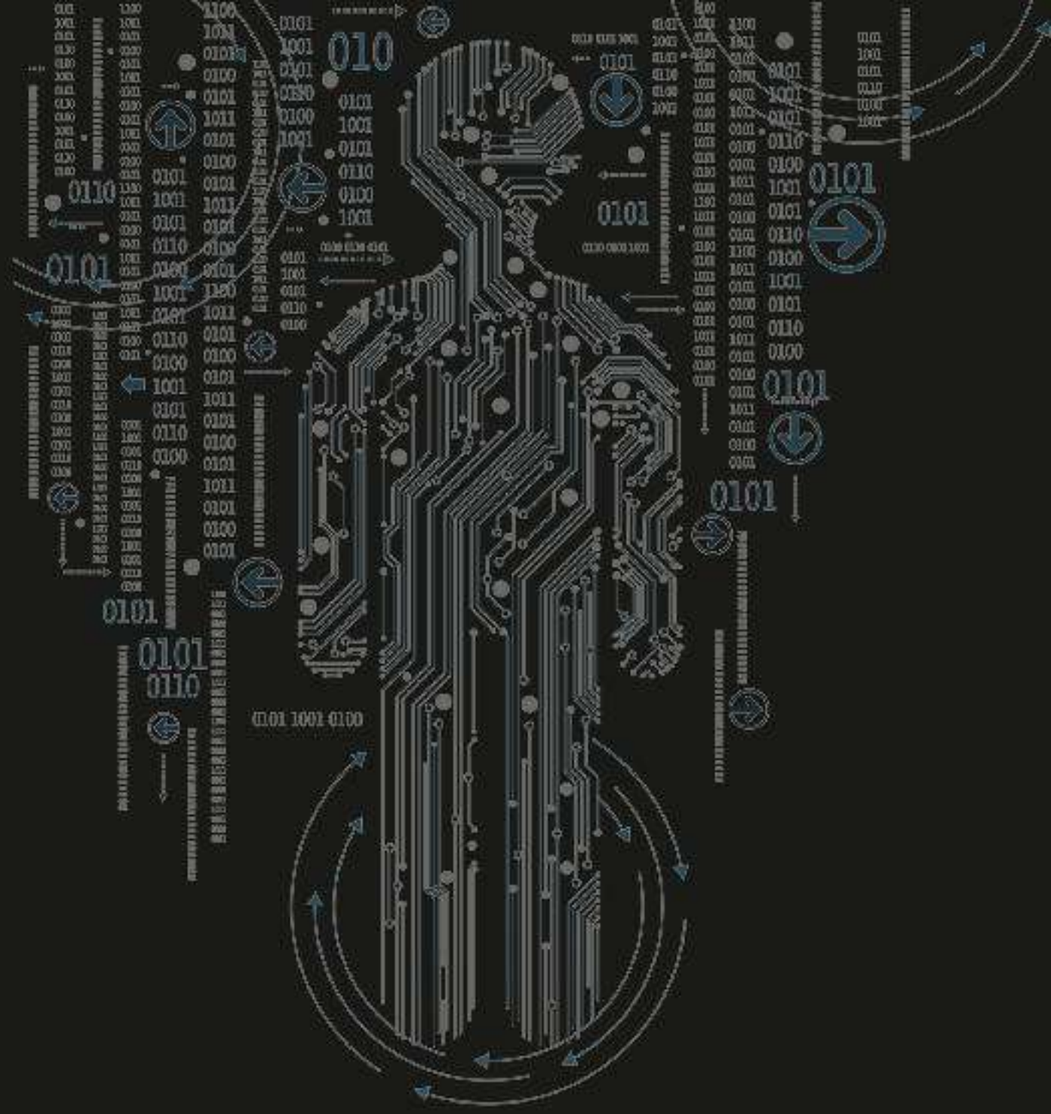
```
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
    {
        printf("Stack is empty\n");
    }
    else
    {
        printf("Printing Stack elements \n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->val);
            ptr = ptr->next;
        }
    }
}
```

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Conversion Of Expressions - 1



① Postfix > Prefix ② < Register / Stack

int a=3, b;

b= ++a + ++a - --a + ++a - ++a;

6 6 6 6 a 6

⇒ 6 + ~~6~~ - ~~6~~ + ~~6~~ - ~~6~~ $\boxed{b=6}$

$\boxed{\cancel{3}\ \cancel{4}\ \cancel{5}\ \cancel{4}\ \cancel{5}\ 6}$

---

int a=3, b; Right to left : Stack.

b= ++a + ++a - --a + ++a - ++a;

5 4 3 2 1

6 5 4 5 4 a

= 6+5-4+5-4

= 16-8 = 8

$\boxed{\cancel{3}\ \cancel{4}\ \cancel{5}\ \cancel{4}\ \cancel{5}\ 6}$

int a=3, b;

b = ++a + ++a - --a + ++a - ++a;

① ② ③ ④ ⑤

L → R

4 + 5 - 4 + 5 - 6

= 14 - 10 = 4

a

$4$4$6

ABOUT ME

- Teaching Experience: 14+ Years
- Achievements: B.Tech, M.Tech, (Ph.D.)
- Achievements:
  - CS Senior Faculty
  - Mentored More Than 30,000+ Students
  - GATE Qualified
  - UGC-NET Qualified
  - SET Qualified
  - CISCO Certified

SATYA NARAYANA KAITEPALLI

Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Conversion Of Expressions - 1

BYJU'S EXAM PREP

Last Class – Quick Revision

- Stack : Last in First out list

Insertion : Push operation

Deletion : Pop operation

Access : Peek operation.

- Stack Implementation :
  a) Arrays
  b) Linked lists
  c) Queues.

Conversion Of Expressions - 1

Q8. What will be the output of below program?

```
#include <stdio.h>
int main(void) {
int a=0,b=1;
if(a,b)
printf(" GOOD ");
else
printf( " BAD ");
return 0;
}
```

if(0,1) Decision making Parameter.

⇒ if(1) = TRUE

A. GOOD

B. BAD

C. No Output

D. Compiler Error

Conversion Of Expressions - 1

Q9. Which of the following is/are the valid syntax of for loop?

1. for(i;j;k)
2. for(i;j;)
3. for(i; ;k)
4. for(; ;k)

for(i; j;)

for(exp1; exp2; exp3)
{
};

for(Initialization; Condition; Inc/Dec)
{
}

A. 1 only

B. 1,2,3

C. 1, 2, 3,4

D. All are Invalid

Conversion Of Expressions - 1

Q10.
```
int main()
{
    int a;
    for(a=printf("%d", a);a>=3;a++)
    {
            //
    }
    return 0;
}
```

The number of times the above loop executes for a=456?

A. 0

B. 1

C. 3

D. Error

5>=3 4>=3 3>=3

a=3

Exp1 Exp2 Exp3

for(a= Pf("%d", a) ; a>=3 ; a++)

{ -- }

a=4

Prints 456

Infinite times

Conversion Of Expressions - 1

Q11. What is the output of below code?

```
void main()
{
int i,j,sum=0;
for(i=0;i<5;i++);
for(j=5;j>0;j/=2)
sum++;
printf("%d",i+j-sum);
}
```

i=0 0<5
i=1 1<5
i=5 5<5 False

j=5 5>0
sum=1
j=5/2=2

i=5
2>0
sum=2
j=2/2=1

j=0
1>0
sum=3
j=1/2=0
0>0 False

2+0−3 = 2

A. 2

B. 4

C. 7

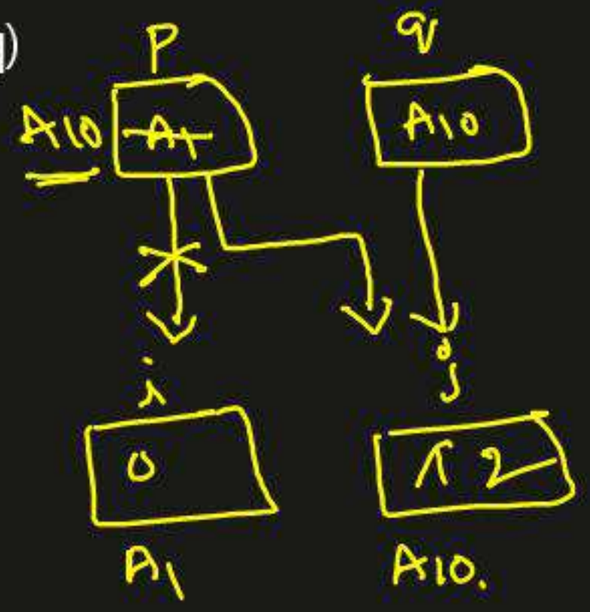
D. 8

Conversion Of Expressions - 1

Q12. What does the following program print?

```
void f(int *p, int *q)
{
  p = q;
  *p = 2;
}
int i = 0, j = 1;
int main()
{
  f(&i, &j);
  printf("%d %d \n", i, j);
  getchar();
  return 0;
}
```

Global Variables



A. 2 2

B. 2 1

C. 0 1

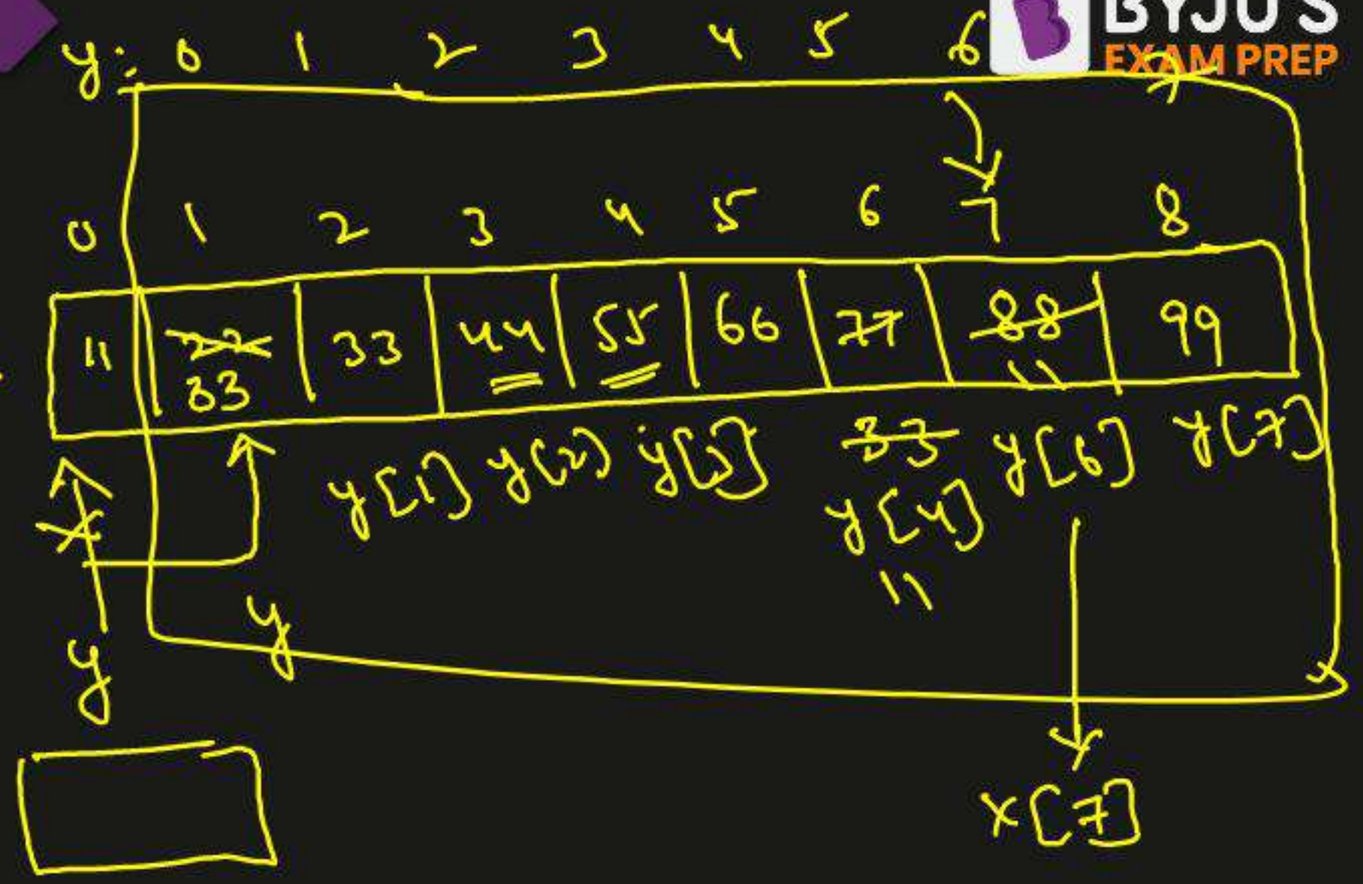
D. 0 2

Conversion Of Expressions - 1

Q13. void main()

```
{
int x[ ]={11,22,33,44,55,66,77,88,99},*y;
y=&x;
y=y+1;
*y=y[1];
y[5]=y[1];
y[6]=y[3] - y[2];
y[7]=y[4] / 3;
printf("%d",x[7]);
}
```

What will be the output?



A. 77

B. 88

C. 11

D. 22

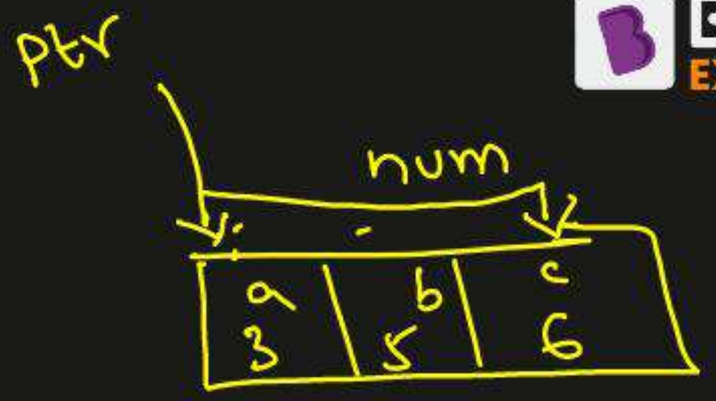
Conversion Of Expressions - 1

Q14. What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        struct node
        {
        int a, b, c;
        };
        struct node num = {3, 5, 6};
        struct node *ptr = & num;
        printf("%d\n", *((int*)ptr + 1 + (3-2)));
        return 0;
}
```

*(ptr + 1 + 1)

*(ptr + 2)



A. 3

B. 5

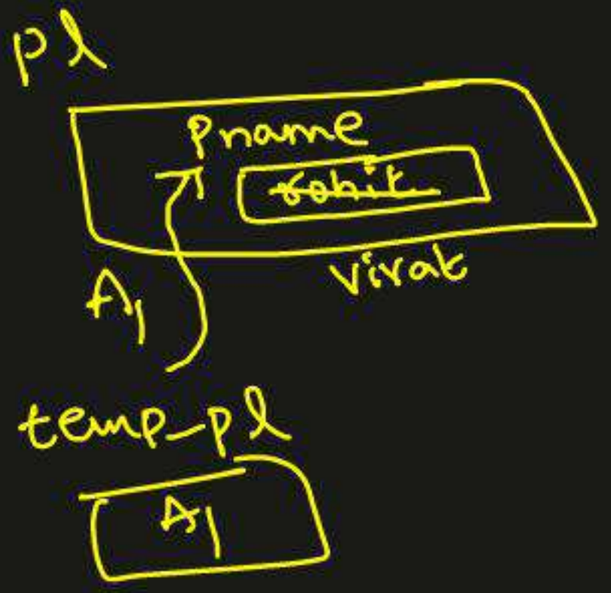
C. Compilation error

D. 6

Conversion Of Expressions - 1

Q15. What will be the output of the C program?

```
#include<stdio.h>
#include<string.h>
struct player
{
    char pname[20];
}pl;
char* play(struct player *temp_pl)
{
    strcpy(temp_pl->pname, "virat");
    return temp_pl->pname;
}
int main()
{
    strcpy(pl.pname, "rohit");
    printf("%s %s", pl.pname, play(&pl));
    return 0;    }
```



virat

Right to left.

A. rohit virat

B. rohit rohit

C. virat rohit

D. virat virat

Conversion Of Expressions - 1

BYJU'S EXAM PREP

Expressions, Conversions

$(+, -, *, /, \wedge$ (Exponentiation))

An Arithmetic Expression can be expressed in either of following ways:

- Infix Expression $a + b$
- Prefix Expression $+ a b$
- Postfix Expression $a b +$

Hence, Possible Conversions Include:

- Infix To Prefix *
- Infix To Postfix *
- Prefix To Postfix
- Prefix To Infix
- Postfix To Prefix
- Postfix To Infix

$2^6 \Rightarrow 2 \wedge 6$

1. $\wedge$ Priority
2. $/, *$
3. $+, -$

$+ - * 2 3 / 4 5 \wedge 6 - 7 8 9$

Applications of stack

1. Recursion
2. Interrupt handling
3. * Conversion of Expressions
4. * Expression evaluation
5. Bracket Balancing

⋮

Conversion Of Expressions - 1

BYJU'S EXAM PREP

Infix To Postfix

To be contd...

1. Push "("onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
   i. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
   ii. Add operator to Stack. [End of If]

BYJU'S EXAM PREP

Infix To Postfix

6. If a right parenthesis is encountered ,then:
   i. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   ii. Remove the left Parenthesis. [End of If] [End of If]
7. END.

Conversion Of Expressions - 1

Infix To Postfix

Example 1 : A+B*(C^D-E)

Conversion Of Expressions - 1

Infix To Postfix

Example 2 : A + B * (C + D)/ F + D * E

Conversion Of Expressions - 1

BYJU'S EXAM PREP

Infix To Prefix

To be Contd...

1. Reverse the given infix expression. (Note: do another reversal only for brackets).
2. Do Infix to postfix expression and get the result.
3. Reverse the result to get the final expression. (prefix expression) .

Conversion Of Expressions - 1

Infix To Prefix

Example 1 : A / B ^ C + D * E - A * C

Conversion Of Expressions - 1

Infix To Prefix

Example 2 : A + B * (C + D)/ F + D * E

Conversion Of Expressions - 1

BYJU'S EXAM PREP

Prefix to Postfix

Step 1: Iterate the given expression from right to left, one character at a time.

Step 2: If the character is operand, push it to the stack.

Step 3: If the character is operator,

a. Pop an operand from the stack, say it's s1.

b. Pop an operand from the stack, say it's s2.

c. perform (s1 s2 operator) and push it to stack.

Step 4: Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.

Step 5: Return the result.

Conversion Of Expressions - 1

Prefix to Postfix

Example 1 -

Prefix Expression : *+A*BC-/DEF

Conversion Of Expressions - 1

Prefix to Postfix

Example 2 -

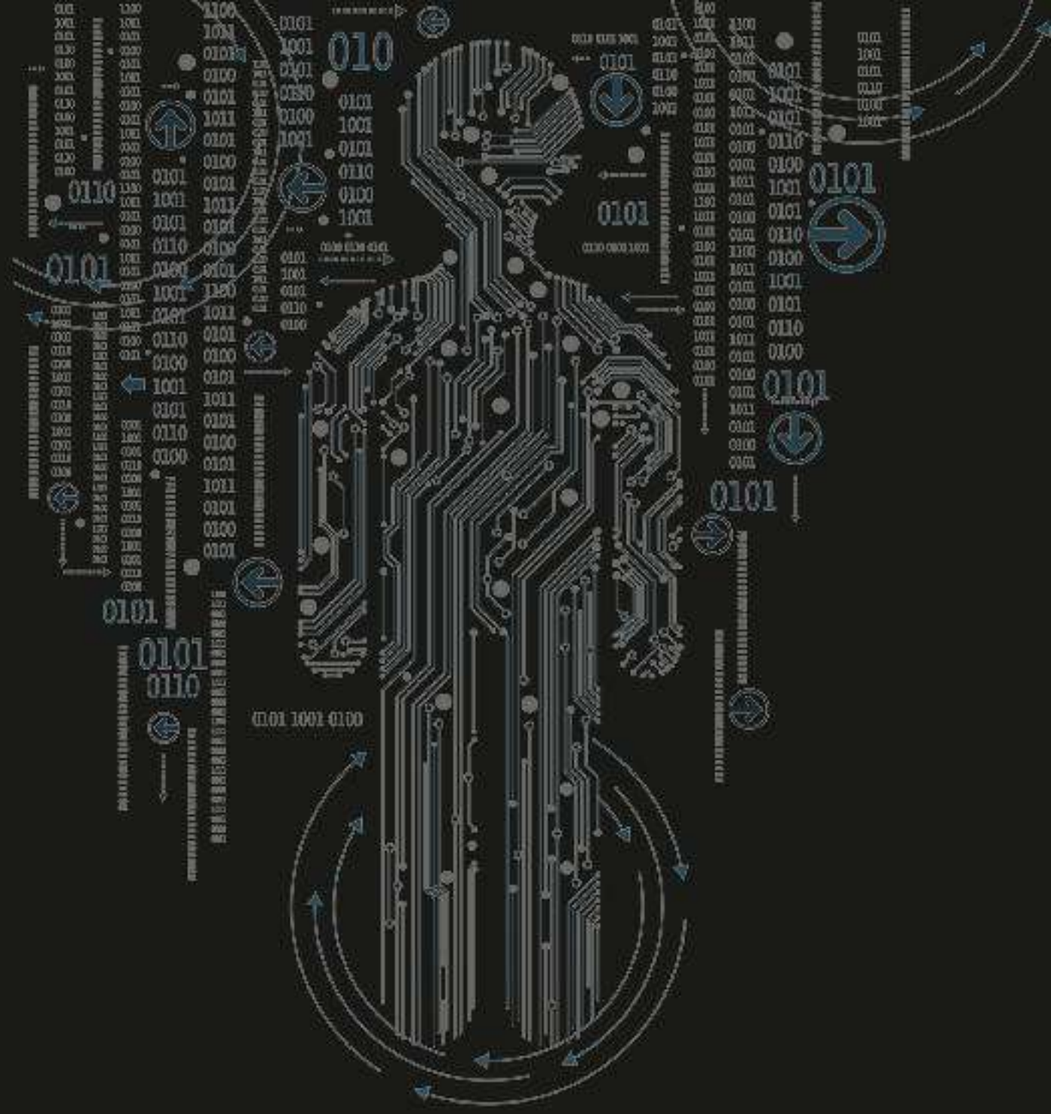
Prefix Expression : *-AB/+CD*XY

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Conversion Of Expressions - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Infix To Postfix

X : infix Expression

Y : Postfix Expression

1. Push "(" onto Stack, and add ")" to the end of X. Can be skipped
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
   i. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than scanned operator.
   ii. Add operator to Stack. [End of If]

Infix To Postfix

6. If a right parenthesis is encountered ,then:
   i. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   ii. Remove the left Parenthesis. [End of If] [End of If]
7. END.

Operator Stack

Infix To Postfix

Scanned operator (*) > Top of stack (+)

Scanned operator (-) < Top of stack (^)

Example 1 : A+B*(C^D-E)

Scanned : A, +, B, *, (, C, ^, D, -, E,), Empty Stack.

Stack :

+	*	(	^	-	)

Y : A B C D ^ E - * +

Infix To Postfix

Example 2 : A + B * (C + D)/ F + D * E

$* == /$

$+ < /$

$/ > +$

$+ == +$

$+ < *$

Scanned Element : A, +, B, *, (, C, +, D,), /, F, +, D, *, E,

Make Empty Stack

Stack :

~~+~~	~~*~~	~~(~~	~~+~~	~~+~~	~~/~~	+	*

Y : ABCD+*F/+DE*+

BYJU'S EXAM PREP

Ex: 3 Infix : $P + Q / R - S * (T - U / V) \wedge W * Z$

Scanned element : P, +, Q, /, R, −, S, *, (, T, −, U, /, V,), ^, W, *, Z

Stack :

+	*	−	*	(	*	*	(	^	*

Y : PQR/+STUV/−W^*Z*−

Top		Scanned	
+	<	/	Push
(Pop) /	>	−	
(Pop) +	==	−	(Push)
−	<	*	(Push)
−	<	/	(Push)
*	<	^	

Top		Scanned	
		*	
(Pop) ^	>	*	
(Pop) *	==	*	(Push)
*	<		

Infix To Prefix

Scan from Right to left (==),) == (

1. Reverse the given infix expression. (Note: do another reversal only for brackets).
2. Do Infix to postfix expression and get the result. (Postfix result)
3. Reverse the result to get the final expression. (prefix expression) .

Infix To Prefix

Example 1 : A / B ^ C + D * E - A * C : C * A - E * D + C ^ B / A

Scanned : C, *, A, -, E, *, D, +, C, ^, B, /, A

Stack :

~~*~~	~~-~~	~~*~~	+	~~A~~	/

Postfix : C A * E D * - C B ^ A / +

Prefix : + / A A B C - * D E * A C

Operator Stack

Infix To Prefix

Reverse

E * D + F / (D + C) * B + A

Example 2 : A + B * (C + D)/ F + D * E

Scanned E, *, D, +, F, /, (, D, +, C,), *, B, +, A

Stack

~~*~~	~~+~~	~~+~~	~~(~~	~~+~~	~~+~~	~~*~~	+

Postfix: ED*FDC+/B*+A+

Prefix : +A+*B/+CDF*DE

Prefix to Postfix → Operand Stack

Step 1: Iterate the given expression from right to left, one character at a time.

Step 2: If the character is operand, push it to the stack.

Step 3: If the character is operator,

a. Pop an operand from the stack, say it's s1.

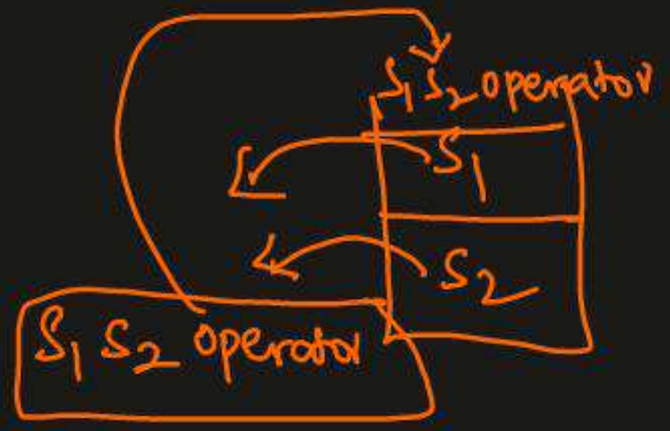
b. Pop an operand from the stack, say it's s2.

c. perform (s1 s2 operator) and push it to stack.

(Represent)

Step 4: Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.

Step 5: Return the result.



Prefix to Postfix

Postfix: ABC*+DE/F-*

Pop result

Example 1 -

Prefix Expression : *+A*BC-/DEF

L ← R

Scanned : F, E, D, /, −, C, B, *, A, +, *

Stack:

~~F~~	~~E~~	~~D~~	~~DEF~~	~~DE/F−~~	~~C~~	~~B~~	~~BC*~~	~~A~~	~~ABC*+~~

S_2 S_1 S_2 S_1 S_2 S_1 S_2 S_1

S_2 S_1 S_2 S_1

ABC*+DE/F−*

Prefix to Postfix

Home work

Example 2 -

Prefix Expression : *-AB/+CD*XY

Conversion Of Expressions - 2

Postfix To Prefix

Iterate the given expression from left to right, one character at a time.

1. If the character is operand, push it to stack.
2. If the character is operator,
   i. Pop operand from the stack, say it's s1.
   ii. Pop operand from the stack, say it's s2.
   iii. Perform (operator s2 s1) and push it to stack.
3. Once the expression iteration is completed, initialize the result string and pop out from the stack and add it to the result.
4. Return the result.

Conversion Of Expressions - 2

Postfix To Prefix

Result : * - A / BC - / DEF

Example 1 :

Postfix Expression : ABC/-DE/F-*

Scanned :- A, B, C, /, -, D, E, /, F, -, *

~~A~~	~~B~~	~~C~~	~~/BC~~	-A/BC	~~D~~	~~E~~	~~/DE~~	~~F~~	-/DEF
S_2	S_2	S_1	S_1	S_2	S_2	S_1	S_2 S_1		S_1

Conversion Of Expressions - 2

Postfix To Prefix

Practice.

Example 2 :

Postfix Expression : abc*d/+ed*-

Conversion Of Expressions - 2

Postfix To Infix

Traverse the postfix expression one by one from left to right and then check for the following cases:

Case 1 – if the operand is found, push it in the stack.

Case 2 – if an operator is found, pop two operands, create an infix expression of the three and push the expression as an operand.

S_1 S_2 S_2 op S_1

If top has 'a', next top is 'b' then express as (b op a)

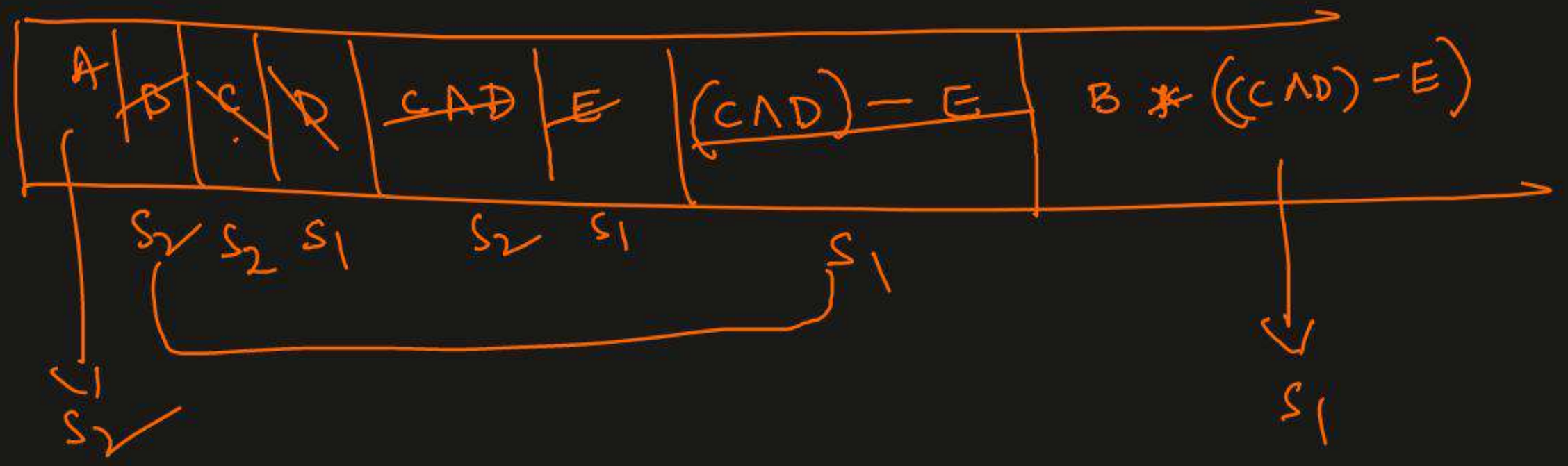
In the end when the stack has only one element left and the traversing is done, pop the top of stack, it is the infix conversion.

Conversion Of Expressions - 2

Postfix To Infix

$A + (B * ((C \wedge D) - E))$

Example 1 : ABCD^E-*+



Conversion Of Expressions - 2

Postfix To Infix

Practice

Example 2 : ABCD+*F/+DE*+

Conversion Of Expressions - 2

Prefix To Infix

Traverse the postfix expression one by one from right to left and then check for the following cases:

Case 1 – if the operand is found, push it in the stack.

Case 2 – if an operator is found, pop two operands, create an infix expression of the three and push the expression as an operand.

S1 S2 S1 op S2

If top has 'a', next top is 'b' then express as (a op b)

In the end when the stack has only one element left and the traversing is done, pop the top of stack, it is the infix conversion.

Conversion Of Expressions - 2

Prefix To Infix

Result: $(A/(B\wedge C)) + ((D*E) - (A*C))$

Example 1 : +/A^BC-*DE*AC

L ← R

~~C~~	~~A~~	~~A*C~~	~~E~~	~~D~~	~~D*E~~	(D*E)-(A*C)	~~C~~	~~B~~	~~(B^C)~~	~~A~~
S_2	S_1	S_2	S_2	S_1	S_1	S_2	S_2	S_1	S_2	S_1

A/(B^C)

S_1

Conversion Of Expressions - 2

Prefix To Infix

Practice

Example 2 : +A+*B/+CDF*DE

Conversion Of Expressions - 2

BYJU'S EXAM PREP

Home work

Q1. The postfix expression for the infix expression

A + B * (C + D) / F + D * E is:

[GATE 1995]

A. AB + CD + *F/D + E*

B. ABCD + *F/+DE*+

C. A *B + CD/F *DE++

D. A + *BCD/F* DE++

Conversion Of Expressions - 2

BYJU'S EXAM PREP

Q2. Which of the following is essential for converting an infix expression to the postfix form efficiently ?

A. An operator stack

B. An operand stack

C. An operand stack and an operator stack

D. A parse tree

[GATE 1997]

Conversion Of Expressions - 2

Q3. The best data structure to check whether an arithmetic expression has balanced parentheses is a ?

[GATE 2004]

A. Queue

B. Stack

C. Tree

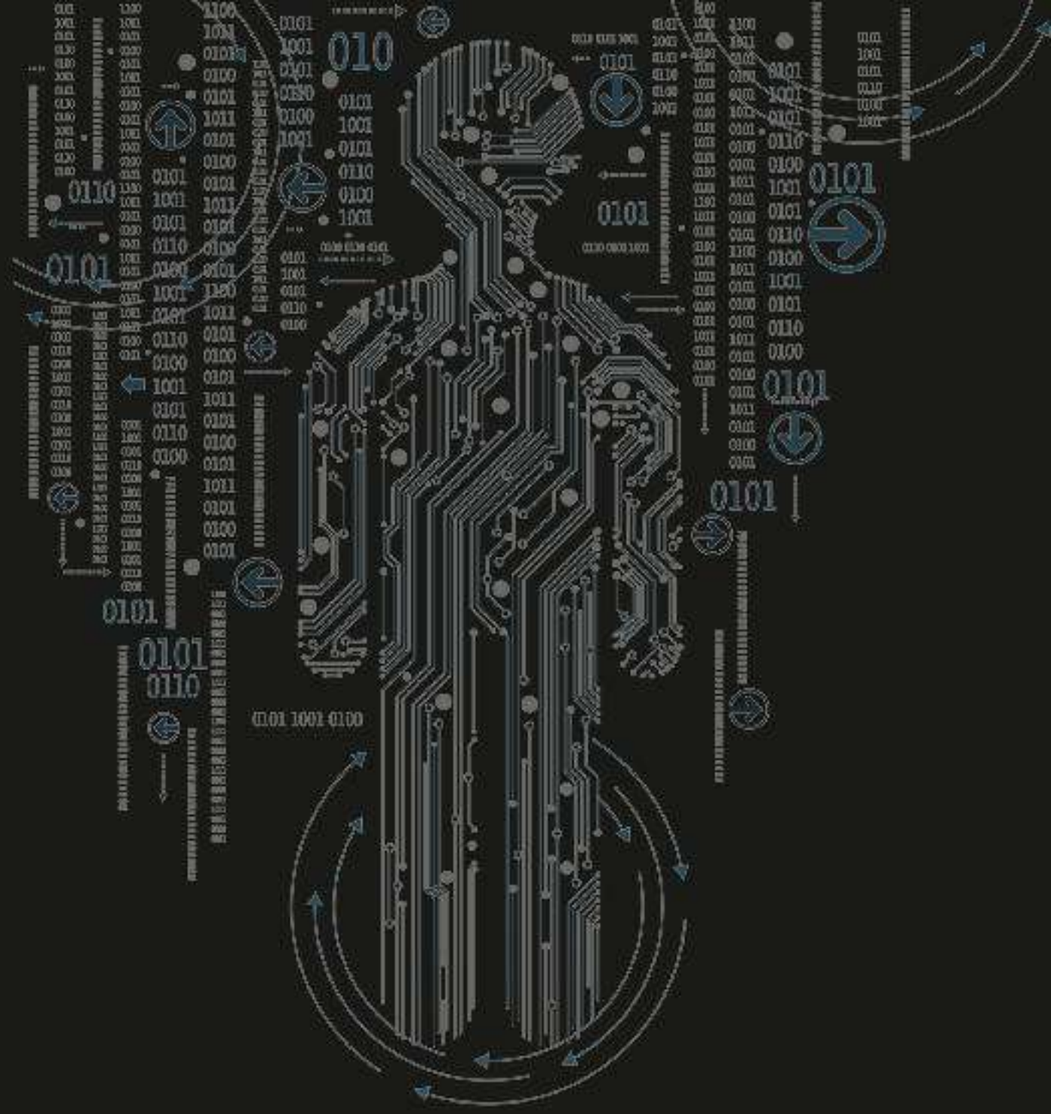
D. List

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Expression Evaluation



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Expression Evaluation

BYJU'S EXAM PREP

Last Class – Quick Revision

- Stack : LIFO list, Push(), Pop(), Peek()
- Implementation : Arrays, Linked lists, Queues
- Stack using Queues, Queue using stack
- Applications of stack : – Conversion of Expressions
  – Expression Evaluation

infix
a+b

* Prefix,
+ab

* Postfix
ab+

BYJU'S EXAM PREP

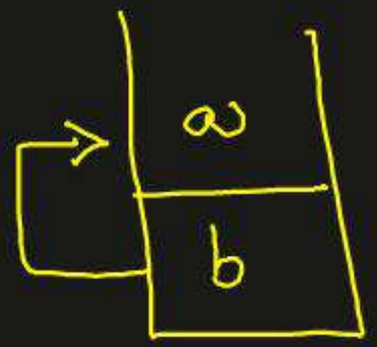
Postfix Expression Evaluation

[Operand Stack]

1. While reading the expression from left to right, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it. (Let top is 'a', Next top is 'b' then perform "b op a")
3. Push back the result of the evaluation. Repeat it till the end of the expression.

4. Pop result.

b op a



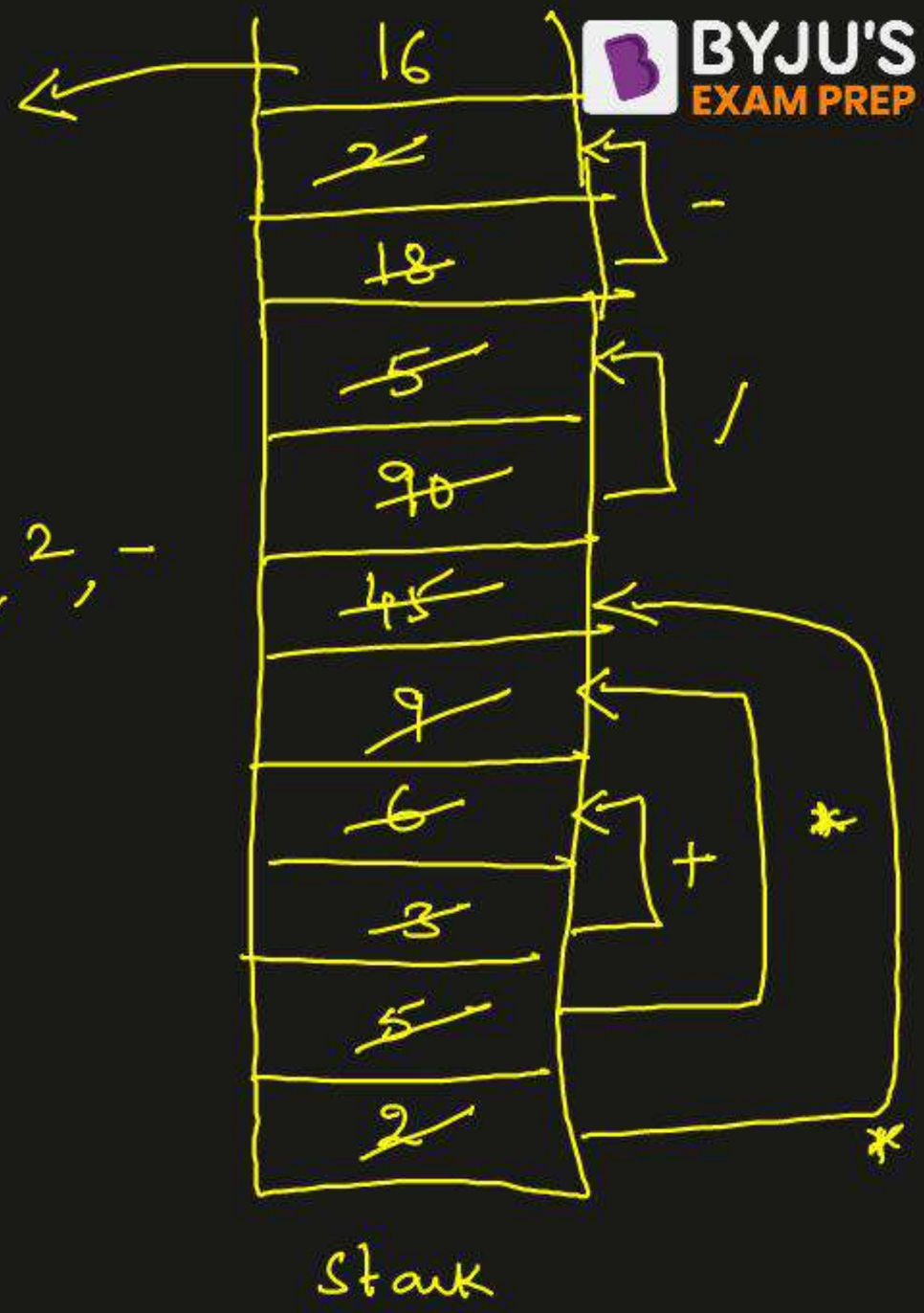
Expression Evaluation

Postfix Expression Evaluation

Example 1 : 2536+**5/2-

Scanned Element : 2, 5, 3, 6, +, *, *, 5, /, 2, -

Result of Postfix Expression = 16



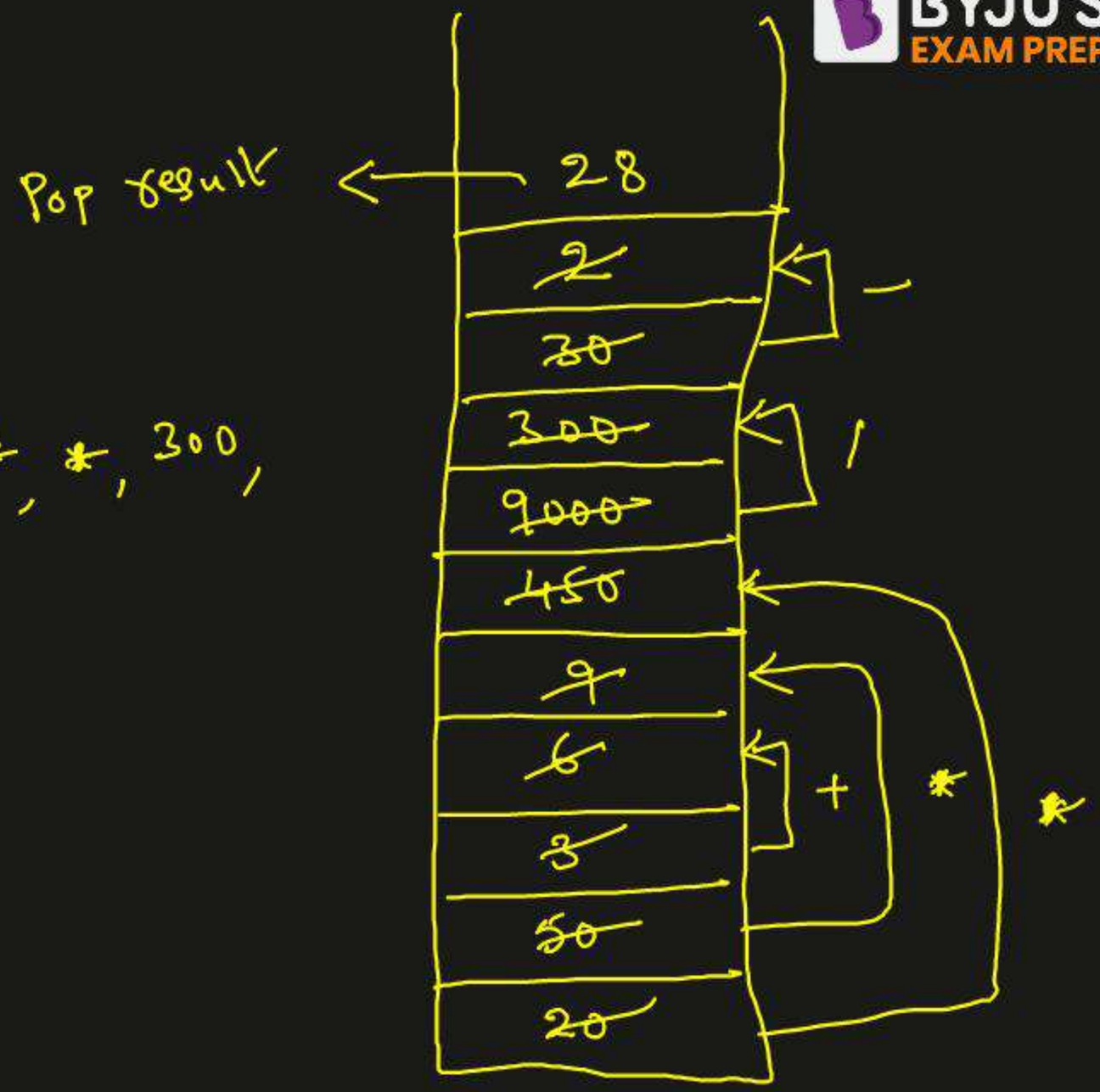
Expression Evaluation

Postfix Expression Evaluation

Example 2 : 20 50 3 6 + * * 300 / 2 -

Scanned Element : 20, 50, 3, 6, +, *, *, 300, /, 2, -

Result = 28



BYJU'S EXAM PREP

Prefix Expression Evaluation

1. While reading the expression from ~~left~~ Right to ~~right~~ Left, push the element in the stack if it is an operand.
2. Pop the two operands from the stack, if the element is an operator and then evaluate it. (Let top is 'a', Next top is 'b' then perform "~~b op a~~") a op b
3. Push back the result of the evaluation. Repeat it till the end of the expression.

4. Pop result.

a op b

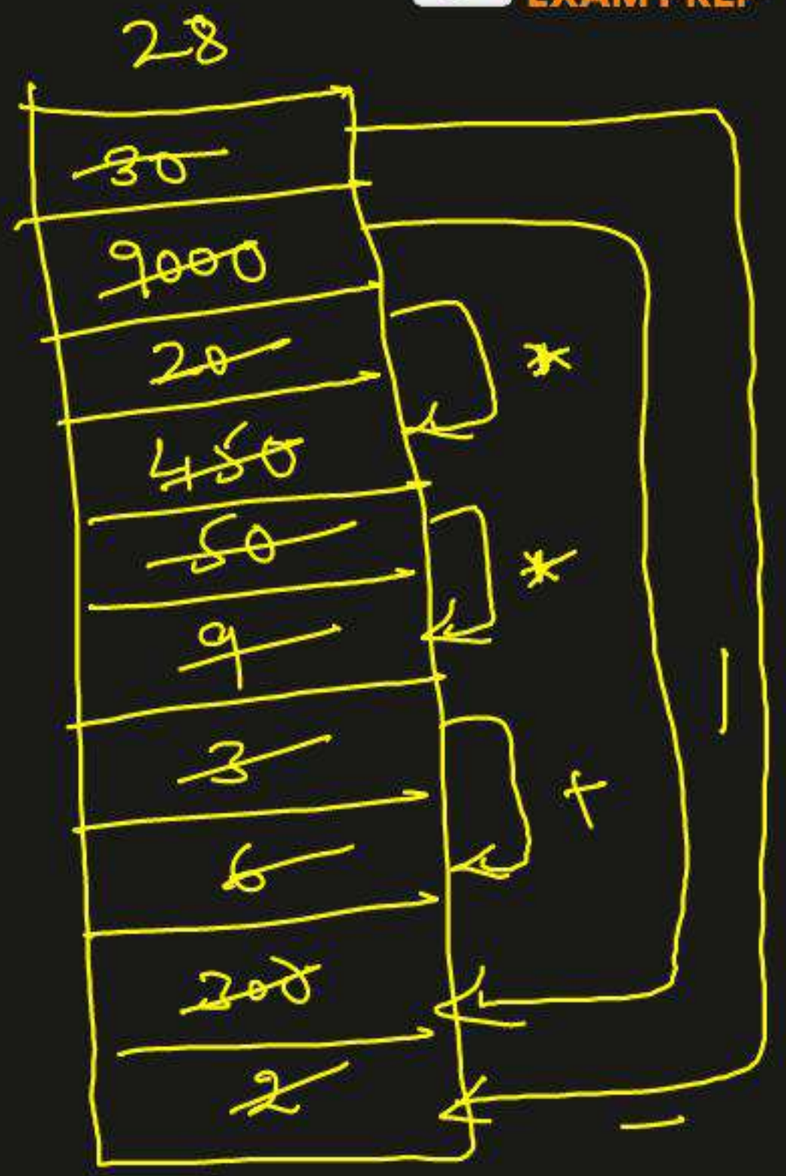
BYJU'S EXAM PREP

Prefix Expression Evaluation

Example 1 : - / * 20 * 50 + 3 6 300 2

Scanned Element : 2, 300, 6, 3, +, 50, *, 20, *, /, -

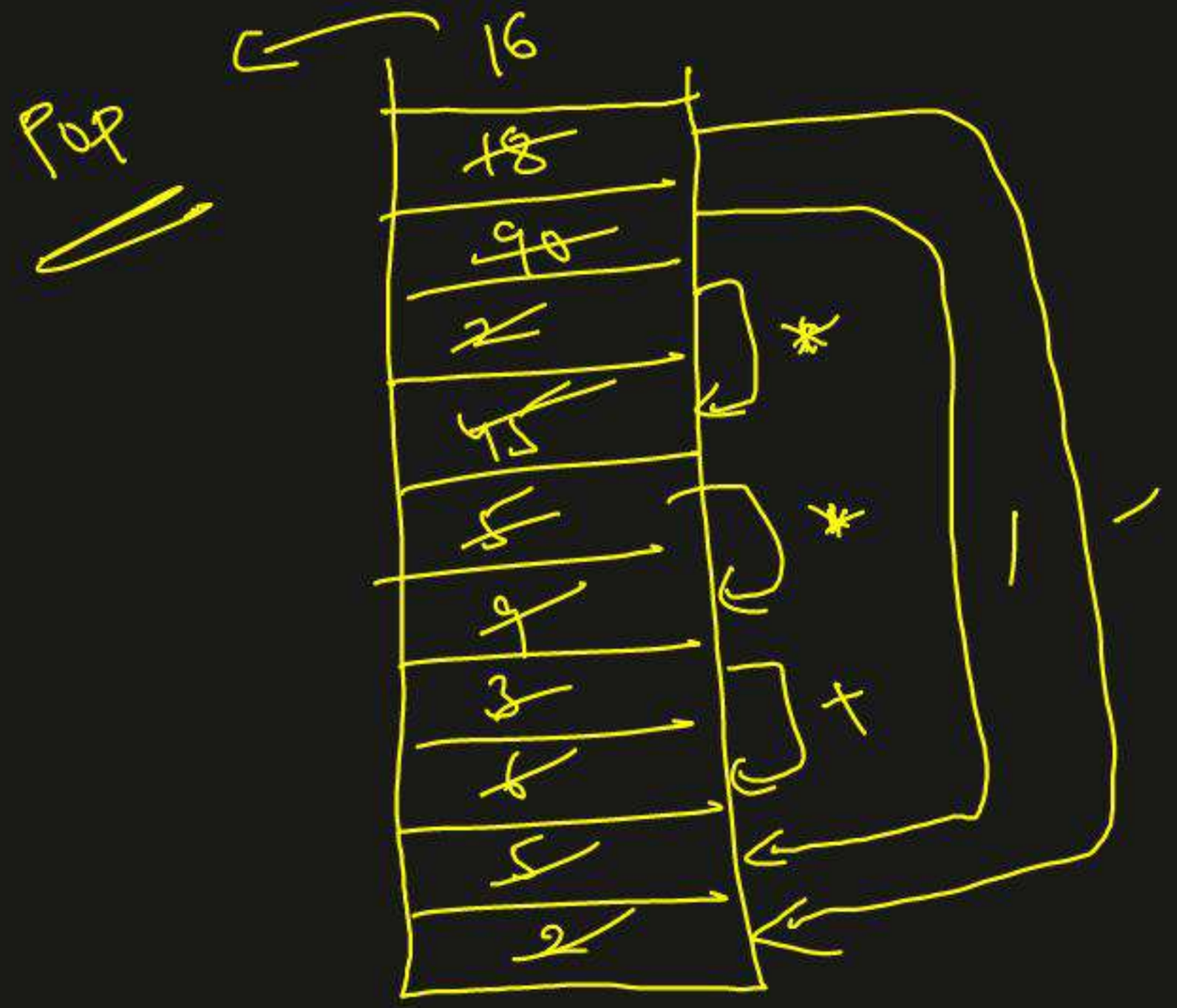
Result : 28



BYJU'S EXAM PREP

Prefix Expression Evaluation

Example 2 : -/*2*5+3652



Result : 16

Expression Evaluation

Q1. The result evaluating the postfix expression 10 5 + 60 6 / * 8 – is

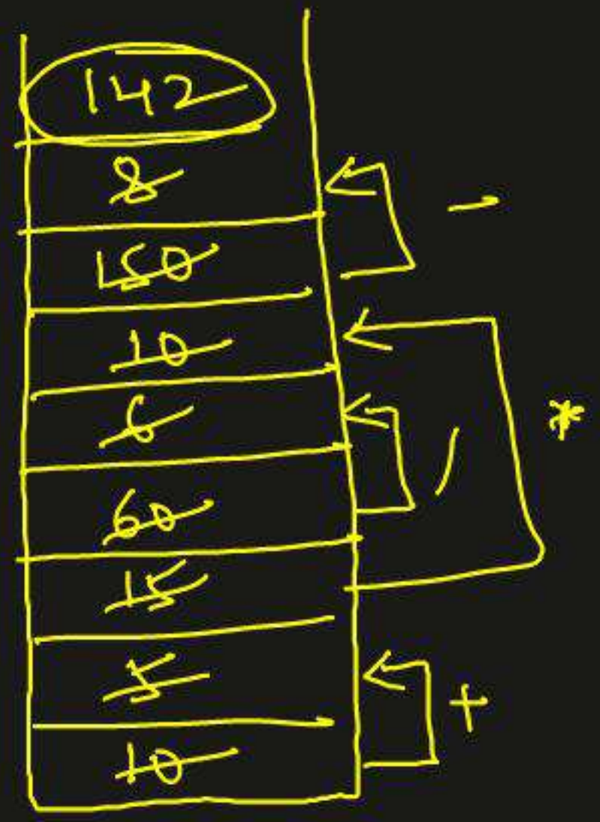
[GATE 2015]

A. 284

B. 213

C. 142

D. 71



BYJU'S EXAM PREP

Q2. A function f defined on stacks of integers satisfies the following properties. f(∅) = 0 and

f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

[GATE 2005]

A. 6

B. 4

C. 3

D. 2

$f(\emptyset) = f(S)$

$\emptyset \neq \neq$

$= \times 3$

$f(\text{push}(S, 2)) = \text{Max}(0, 0) + (2) = 0 + 2 = 2$

$f(\text{push}(S, -3)) = \text{Max}(2, 0) + (-3) = 2 + (-3) = -1$

$f(\text{push}(S, 2)) = \text{Max}(-1, 0) + (2) = 0 + 2 = 2$

$f(\text{push}(S, -1)) = \text{Max}(2, 0) + (-1) = 2 - 1 = 1$

$f(\text{push}(S, 2)) = \text{Max}(1, 0) + 2 = 1 + 2 = 3$

Expression Evaluation

Q3. The following postfix expression with single digit operands is evaluated using a stack:

8 2 3 ^ / 2 3 * + 5 1 * -

Note that ^ is the exponentiation operator. The top two elements of the stack after the first * is evaluated are:

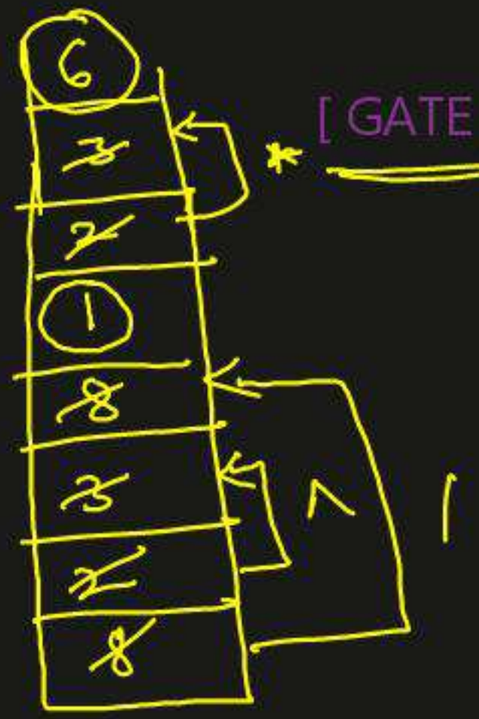
[GATE 2007]

A. 6, 1

B. 5, 7

C. 3, 2

D. 1, 5



Result after evaluation : 2

Expression Evaluation

Q4. The following sequence of operations is performed on stack:

PUSH (10),PUSH (20),POP,PUSH (10),PUSH (20),POP,POP,POP,PUSH (20),POP

The sequence of the value popped out is :

[GATE 1991]

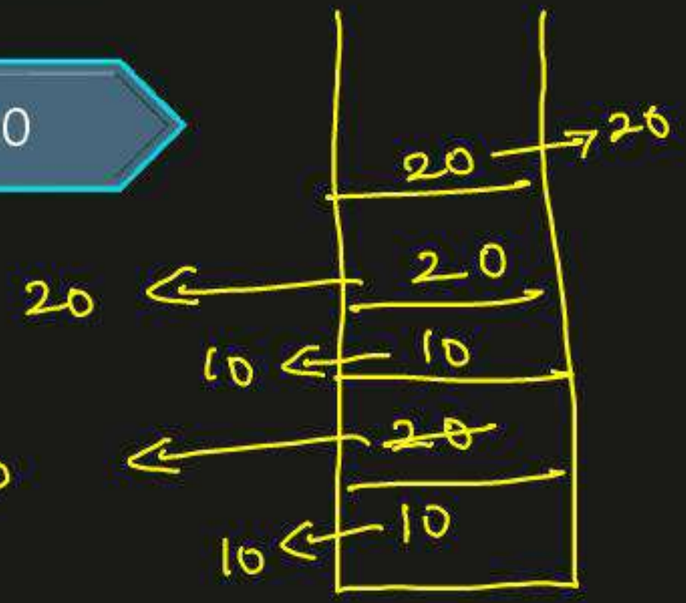
A. 20,10,20,10,20

B. 20,20,10,10,20

C. 10,20,20,10,20

D. 20,20,10,20,10

20,20,10,10,20



BYJU'S EXAM PREP

Q5. Which of the following permutations can be obtained in the output (in the same order) using a stack assuming that the input is the sequence 1, 2, 3, 4, 5 in that order ?

[GATE 1994]

A. 3, 4, 5, 1, 2

B. 3, 4, 5, 2, 1

C. 1, 5, 2, 3, 4

D. 5, 4, 3, 1, 2

4, 3, 2

Push(1)
Push(2)
Push(3) Pop — 3
Push(4) Pop — 4
Push(5) Pop — 5

Pop — 2
Pop — 1

Expression Evaluation

BYJU'S EXAM PREP

Consider the following program:

```
#define funct(x) x*x+x
int main ()
{
int x;
x = 36+funct (5) *funct (3);
printf ("%d", x);
return 0;
}
```

What will be the about of the above program?

A.73 B.396

C.109 D.360

As it is Substitution

$x = 36 + \underbrace{5*5+5}_{25} * \underbrace{3*3+3}_{45}$

wrong

$5*5+5 = 30$

$3*3+3 = 12$

$36+30*12 = 396$

① Evaluation
② Substitution } wrong order.

Correct

① $5*5 = 25$

② $5*3*3 = 45$

③ $36+25+45+3$

Ans: 109

Expression Evaluation

7:30 AM

← Quiz: Pointers

Q.3

Consider the following C program:

```
#include

void f(int x, int *p)

{

    *p =x;

    x=10;

}
```

x: ~~20~~ 10

p: A10

```
int main ()

{

    int a=5, b=6;

    int *p = &a, **q;

    *p = 20; q = &p;

    f(a, &b);

    *q = &b;

    *p = 30;

    printf("%d", b);

}
```

a: ~~5~~ 20 A1

b: ~~6~~ ~~20~~ 30 A10

p: ~~A1~~ A10 A100

q: A100

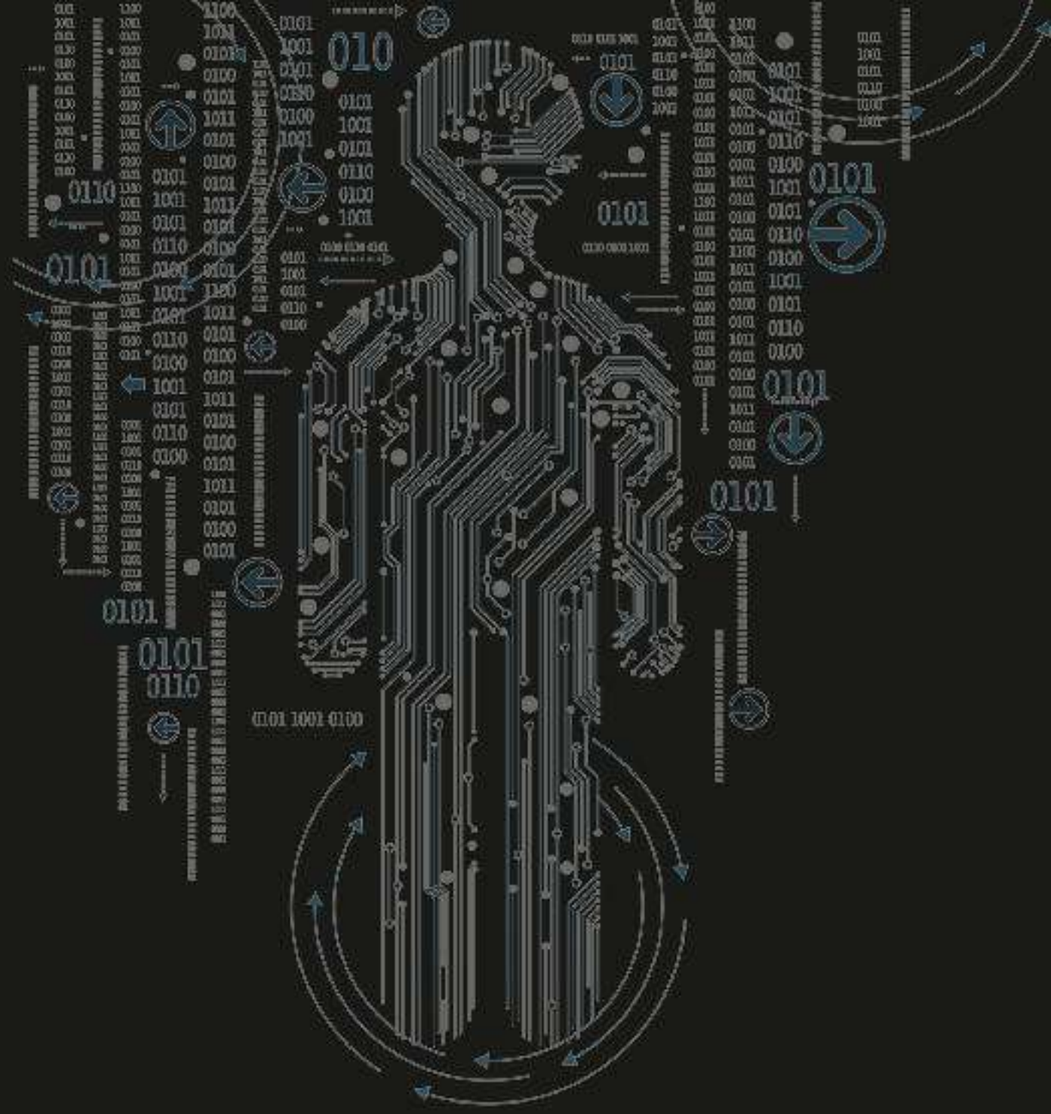
Ans : 30

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Queues - 1



1. Tomorrow Morning to Sunday Morning Rescheduled classes @ 10AM

2. From Next Subject – Algorithms onwards

1 class – 2 hours duration

classes – only weekdays.

Algorithms class – 6pm to 8pm.

Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Conversion of Expressions

- infix to Postfix * * *
- infix to Prefix *

→ Operator Stack

- Postfix to Prefix
- Prefix to Postfix
- Postfix to infix
- Prefix to infix

→ Operand Stack.

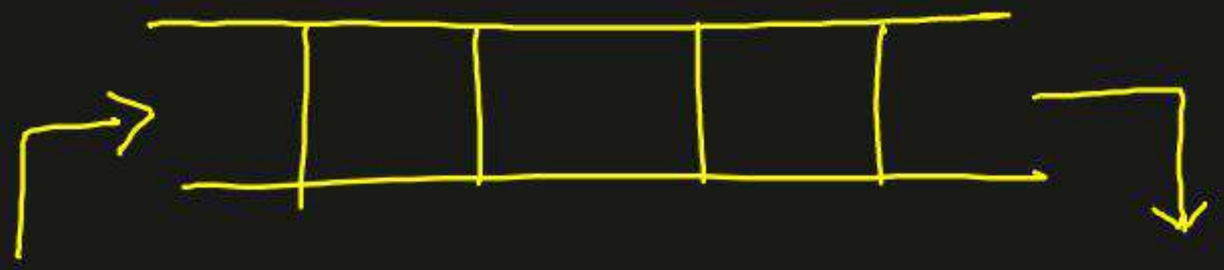
BYJU'S EXAM PREP

Queue

A queue in C is basically a linear data structure to store and manipulate the data elements. It follows the order of First In First Out (FIFO).

In queues, the first element entered into the array is the first element to be removed from the array.

A queue is open at both ends. One end is provided for the insertion of data and the other end for the deletion of data.



BYJU'S EXAM PREP

Operations On Queues

A queue being an Abstract Data Structure provides the following operations for manipulation on the data elements:

- isEmpty(): To check if the queue is empty. : underflow
- isFull(): To check whether the queue is full or not. : overflow
- dequeue(): Removes the element from the frontal side of the queue.
- enqueue(): It inserts elements to the end of the queue.
- Front: Pointer element responsible for fetching the first element from the queue.
- Rear: Pointer element responsible for fetching the last element from the queue.

BYJU'S EXAM PREP

Types Of Queues

- Simple Queue
- Circular Queue

None

- Double Ended Queue (DEQUE) (or) DEQUEUE*
- Priority Queue

Queues - 2

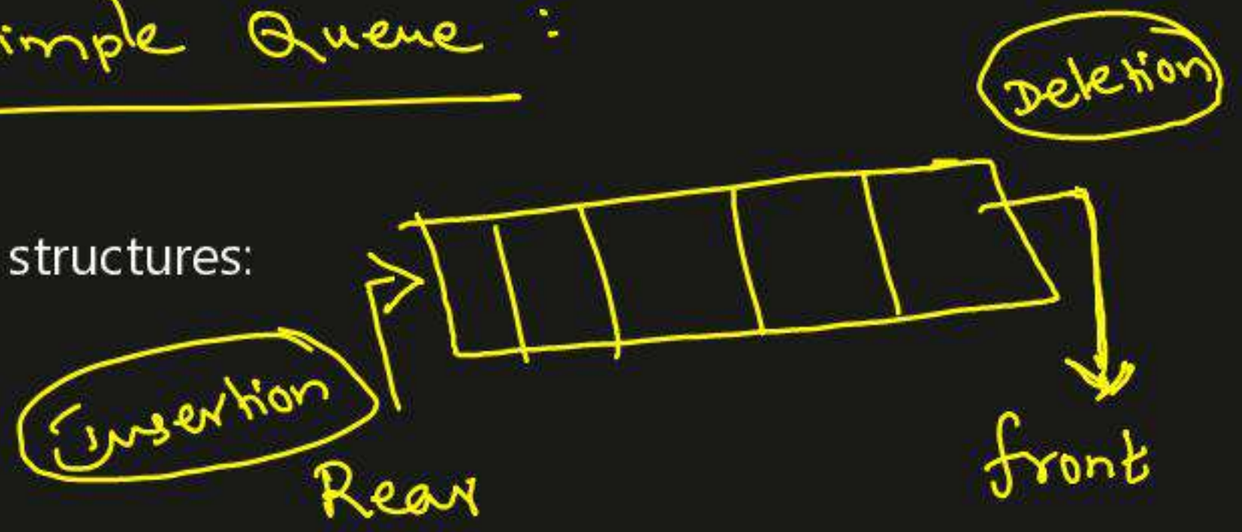
BYJU'S EXAM PREP

Queue Implementation

Queues can be implemented using either of 3 data structures:

- Array ✓
- Linked Lists ✓
- Stack ✓

Simple Queue :



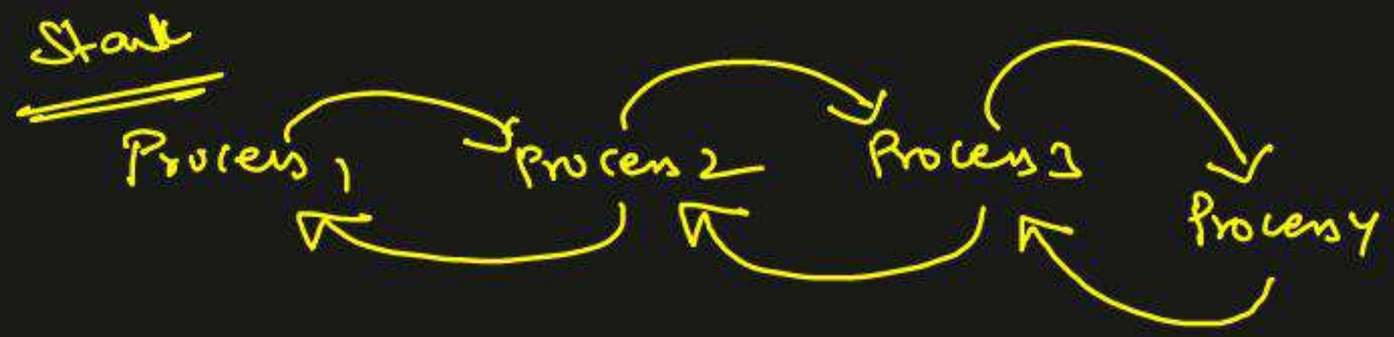
Insertion always from Rear End.

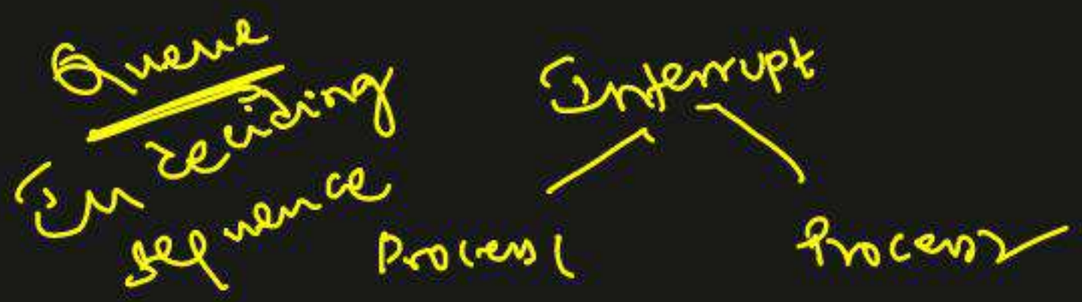
Deletion always from front End.

BYJU'S EXAM PREP

Applications Of Simple Queue

- Process scheduling,
- Disk scheduling,
- Memory management,
- IO buffer,
- Call center phone systems, and
- Interrupt handling.





BYJU'S EXAM PREP

Queue Using Arrays

```
void enQueue(int value){
  if(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(front == -1)
        front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInsertion success!!!");
  }
}
```

Let SIZE = 5 queue[5] array.

Initially Queue is Empty.

front == rear = -1

	rear			rear
rear = 0	1	2	3	4
10	20	30	40	50

queue

front

enqueue (10); value = 10

if (rear == size-1) -1 == 4 F

else

rear++; rear = 0

queue [0] = 10;

enqueue (20) r = 1

(0 == 4) F

0 == -1 False Queue[1] = 20

enqueue [30] rear = 2

enqueue [40] rear = 3

enqueue [50] rear = 4

Now, enqueue(60)

4 == 4 True

Queue is Full.

BYJU'S EXAM PREP

Queue Using Arrays

```
void deQueue(){
  if(front >rear){
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
    front=rear=-1;
  }
  else{
    printf("\nDeleted : %d", queue[front]);
    front++;
  }
}
```

|| (front == -1))

Now, dequeue(); 5 > 4 True Queue is Empty. f = r = -1

f=5, r=4

Now, dequeue(); again (-1 > -1) False Queue[-1]

Let, Queue be

0	1	2	3	4
~~10~~	~~20~~	~~30~~	~~40~~	~~50~~

front rear front = 5

dequeue(); 0 > 4 False

Deleted : Queue[0] ; 10

front = 1;

dequeue(); 1 > 4 False

Deleted : Queue[1] : 20

front = 2;

dequeue(); front = 3

dequeue(); front = 4

dequeue(); front = 5

Let, Queue is Empty. front = NULL
rear == NULL

Simple

Queue Using Linked List

```
void insert(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
```

Overflow Condition is Neglected (Assume Enough Memory space).

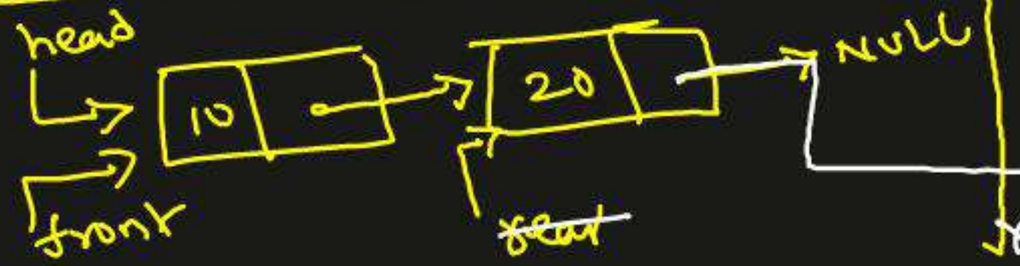
head = New Node;

```
Struct Node
{ int data;
Struct Node *next;
};
```

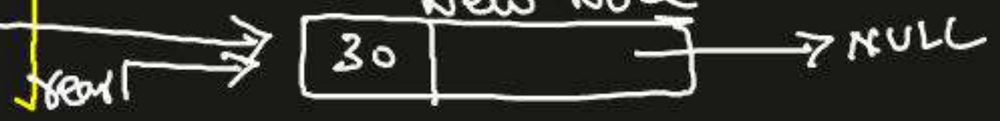
insert(10);

insert(20);

if(f = NULL) False



insert(30);



BYJU'S EXAM PREP

Now, f, h → 10 → 20 → 30 → 40 → NULL

f, h f, h r, f, h f, h

Queue Using Linked List

```
void delete()
{
  if(front == NULL){
    printf("\nQueue is Empty!!!\n");
    rear=NULL;
  }
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
```

, head = front;

10, 20

if (f == NULL) rear = NULL;

delete();

(f == NULL) False

temp

delete();

if (f == NULL) False

temp

Delete(); Delete(); Delete();

(f == NULL) True

Queue is Empty, rear == NULL;

Drawback of Simple Queue, Let SIZE = 5

→ A Queue Can be Made full only once. From Then onwards, though there is enough space, overflow occurs. Memory wastage.

Let, initially $f = r = -1$ Q is empty.

enqueue(10); $f=0, r=0$

enqueue(20); $f=0, r=1$

enqueue(30); $f=0, r=2$

enqueue(40); $f=0, r=3$

enqueue(50); $f=0, r=4$

enqueue(60); overflow $r(4) == SIZE-1\ (4)$ True,

dequeue(); $f=1, r=4$

dequeue(); $f=2, r=4$

enqueue(60); overflow $rear(4) == SIZE-1\ (4)$ True

front rear

0	1	2	3	4
10	20	30	40	50

front rear

0	1	2	3	4
		30	40	50

BYJU'S EXAM PREP

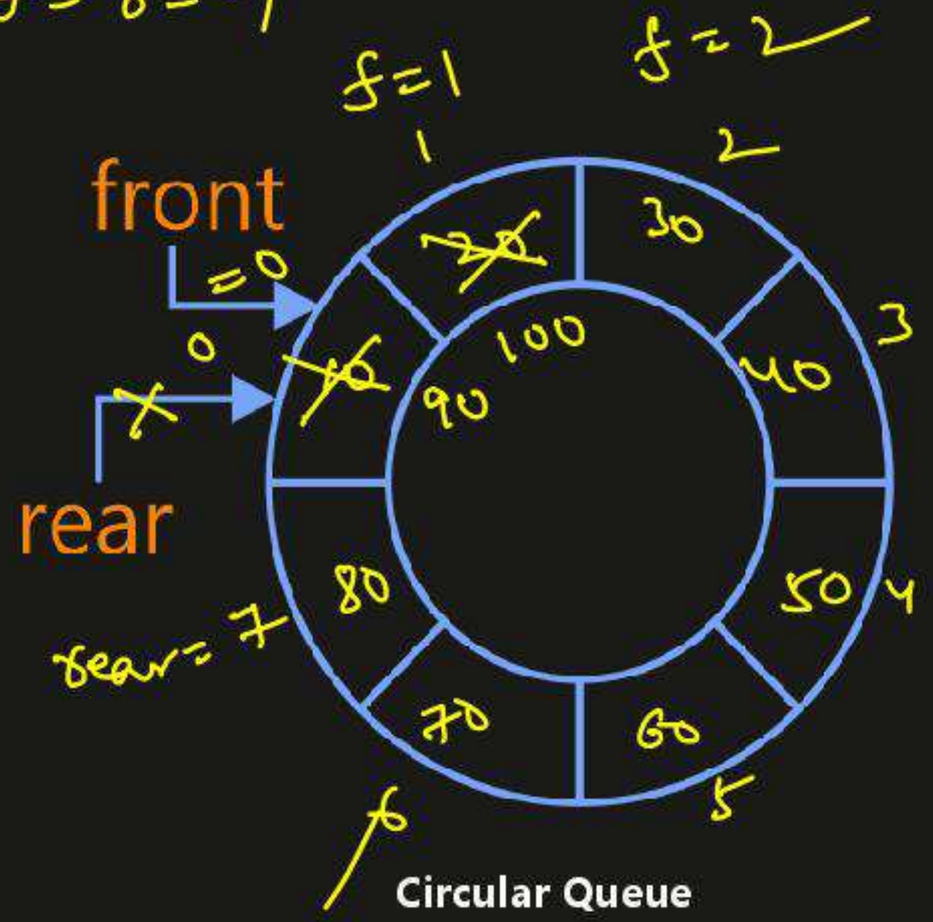
let SIZE = 8

Circular Queue

f = r = -1

enque(10)
enque(20)
30
40
50
60
70
80

dequeue();
dequeue();
enque(90);
enque(100);



Circular Queue

r = (r+1) % SIZE
f = (f+1) % SIZE
→ Modulo Increment

BYJU'S EXAM PREP

Circular Queue

```
void enQueue(int element) {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
   CQueue[rear] = element;
    printf("\n Inserted -> %d", element);
  }
}
```

[f > r]

Circular Queue

Using array

```
int deQueue() {
  int element;
  if (front == -1) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
    // Q has only one element, so we reset the
    // queue after dequeing it. ?
    else {
      front = (front + 1) % SIZE;
    }
    printf("\n Deleted element -> %d \n", element);
    return (element);
  }
}
```

BYJU'S EXAM PREP

Circular Queue Using Linked List

```
void enqueue(int d) //Insert elements in Queue
{
        struct node* n;
        n = (struct node*)malloc(sizeof(struct node));
        n->data = d;
        n->next = NULL;
        if((r==NULL)&&(f==NULL))
        {
                f = r = n;
                r->next = f;
        }
        else
        {
                r->next = n;
                r = n;
                n->next = f;
        }
}
```

BYJU'S EXAM PREP

Circular Queue Using Linked List

```
void dequeue() // Delete an element from Queue
{
        struct node* t;
        t = f;
        if((f==NULL)&&(r==NULL))
                printf("\nQueue is Empty");
        else if(f == r){
                f = r = NULL;
                free(t);
        }
        else{
                f = f->next;
                r->next = f;
                free(t);
        }


}
```

BYJU'S EXAM PREP

Applications Of CQueue

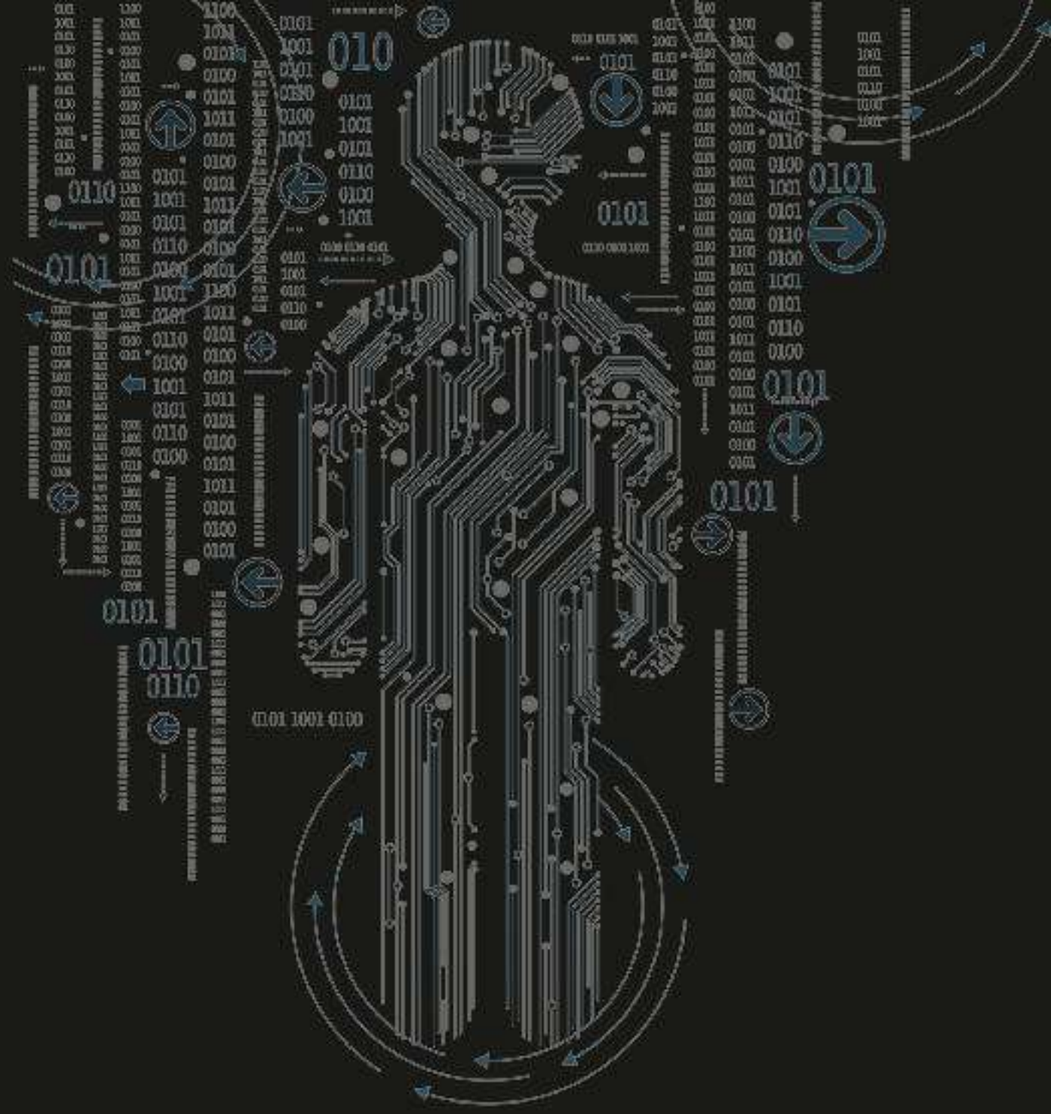
- Memory management: circular queue is used in memory management.
- Process Scheduling: A CPU uses a queue to schedule processes. [RR]
- Traffic Systems: Queues are also used in traffic systems.

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Queues - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

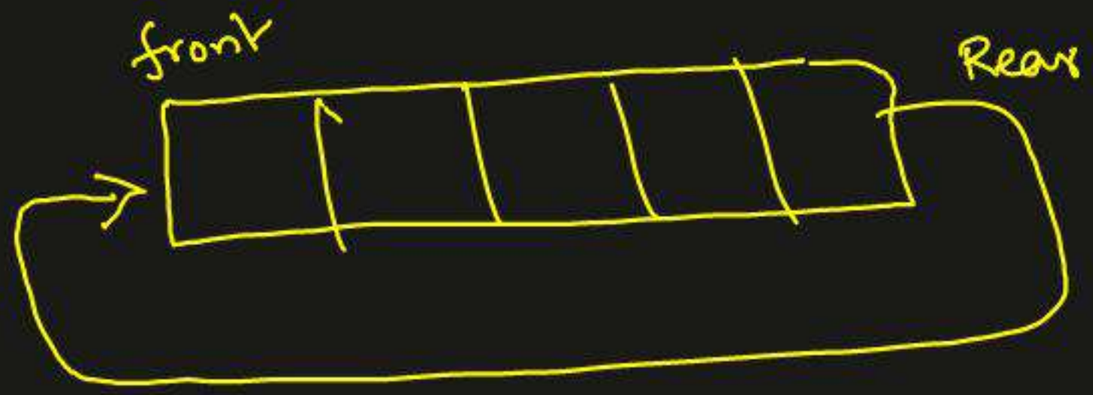
Last Class – Quick Revision

- Simple Queue :

Insertion : enqueue( ) Deletion: dequeue( )

- Drawback : Memory Wastage

- Circular Queue:



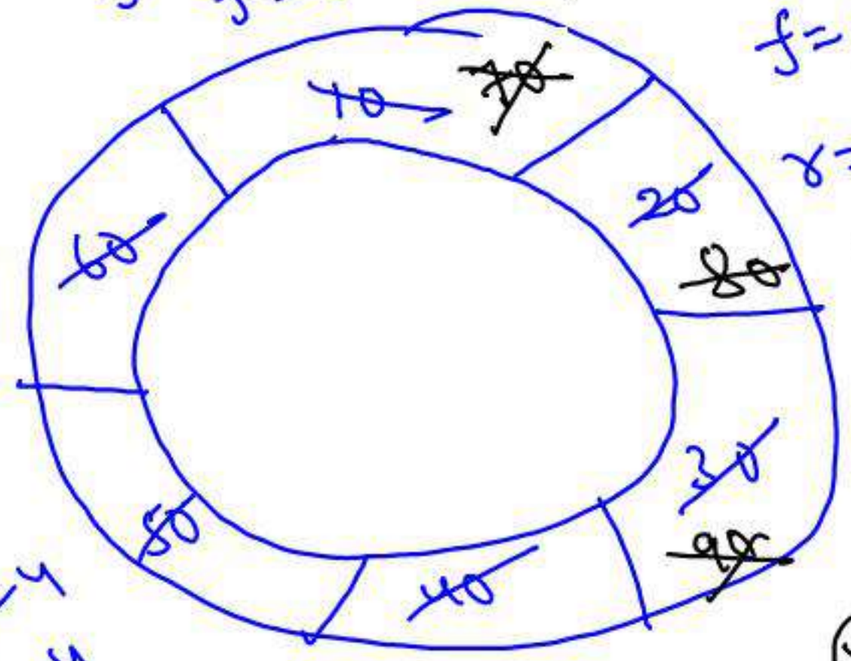
$R = (R+1) \% size$ $f = (f+1) \% size$

$f = r = -1$ Circular queue is Empty.

Size = 6

$f = (r+1) \% size$

FULL



$f = 0$

$f = r = 0$

$r = 0$

$f = 1$ $(f+1) \% size$

$r = 1$ $(0+1) \% 6$

$(r+1) \% size$

$r = 1$

$f = 1$

$r = 2$

$f = 2$

$f = 2$

$r = 3$

$f = 3$

$f = 3$

$r = 4$

$f = 4$

$r = 5$

$f = 5$

$(f == r)$ Empty.

$(f = -1)$ Empty

Deletion

- Delete
- $f = (f+1) \% size$

Insertion

- $r = (r+1) \% size$
- Insert

enqueue(70)

enqueue(80)

enqueue(90)

dequeue();

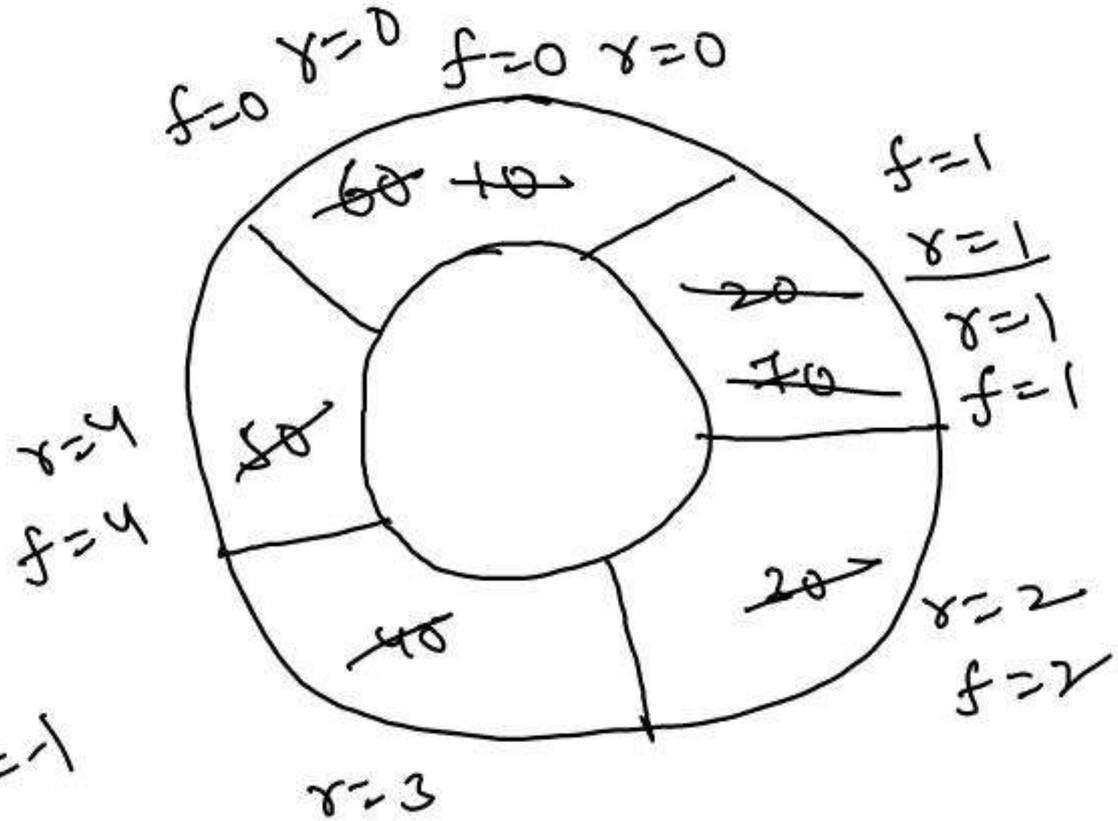
dequeue();

dequeue();

SIZE = 5

f = r = -1 Empty

f == r



Delete

if (f == r) r = f = -1
{ CQ is Empty }
, else f = (f+1) % SIZE

Full

f == (r+1) % SIZE

0 == (4+1) % 5

0 == 5 % 5

0 == 0

1 = (0+1) % 5

1 == 1

2 = (1+1) % 5

2 == 2

BYJU'S EXAM PREP

SQ, CQ : Insertion : Rear
Deletion : front

Double Ended Queue

Deque or Double Ended Queue is a type of queue in which insertion and removal of elements can either be performed from the front or the rear. Thus, it does not follow FIFO rule (First In First Out).

o/p sequence : 50, 40, 30, 20, 10 i/p sequence : 10, 20, 30, 40, 50 (Stack)

at insert front

enqueue(10)
enqueue 20)
enqueue(30)
enqueue(40)
enqueue(50)

front
dequeue() : 50
dequeue() : 40
dequeue() : 30
dequeue() : 20
dequeue() : 10

Double Ended Queue

0	1	2	3	4
10	~~20~~	~~30~~	~~40~~	~~50~~

f
r = -1, f = -1
f = 0
f = 1, r = 1
f = 2, r = 2
f = 3, r = 3
f = 4

Insert From Front*, Delete From Rear*

$f = r = -1$

enqueue(10)

enqueue(20)

enqueue(30)

enqueue(40)

enqueue(50)

0	1	2	3	4
~~10~~	~~20~~	~~30~~	~~40~~	~~50~~

$f=0$ $f=1$ $f=2$ $f=3$ $f=4$ $r=5$

~~$r=0$~~ $r=1$ $r=2$ $r=3$ $r=4$

dequeue(); 10

dequeue(); 20

dequeue(); 30

dequeue(); 40

dequeue(); 50

$f = r = -1$

i/p sequence : 10, 20, 30, 40, 50

o/p sequence : 10, 20, 30, 40, 50

} Simple Queue

Delete from front/Rear, Insert from front/Rear (STACK)

$f = r = -1$

enqueue(10)
enqueue(20)
" (30)
" (40)
" (50)

i/p

r++;
f++;

0	1	2	3	4
~~10~~	~~20~~	~~30~~	~~40~~	~~50~~
r=0	r=1	r=2	r=3	r=4

r=-1 r=0 r=1 r=2 r=3

dequeue(): 50, 40, 30, 20, 10 o/p

r--;
f--;

NOTE: In dequeue,

if insertion/Deletion from Same Ends - Stack

if insertion/Deletion from diff ends - Queue

front: f++; | rear: r++; | rear: r++; | front: f++;

BYJU'S EXAM PREP

Types Of Deque

Types of Deque :

- Input Restricted Deque

  In this deque, input is restricted at a single end but allows deletion at both the ends.

- Output Restricted Deque

  In this deque, output is restricted at a single end but allows insertion at both the ends.

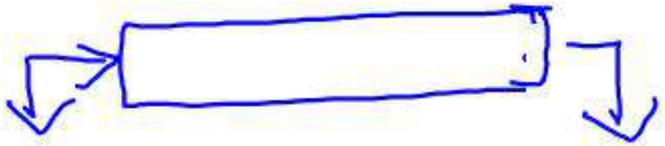
BYJU'S EXAM PREP

Operations

- In a circular array, if the array is full, we start from the beginning.
- But in a linear array implementation, if the array is full, no more elements can be inserted. If the array is full, "overflow message" is thrown.
- Take an array (deque) of size n.
- Set two pointers at the first position and set front = -1 and rear = 0.

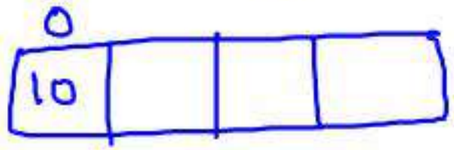
Input-restricted dequeue

$f = r = -1$



Let insert from rear end only but delete from both ends.

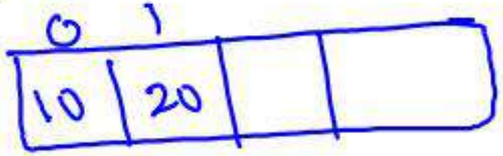
enqueue(10);



$f = r$

$f = 0, r = 0$

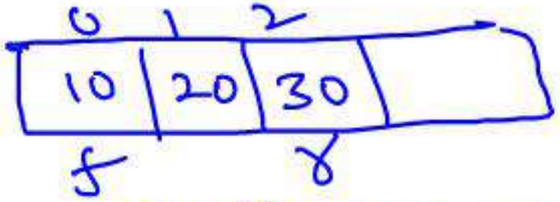
enqueue(20);



$f = 0$ $r = 1$

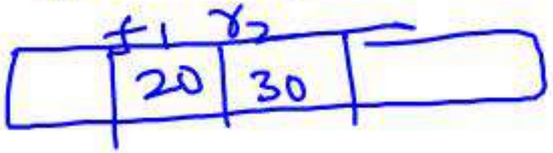
$f = 0, r = 1$

enqueue(30);



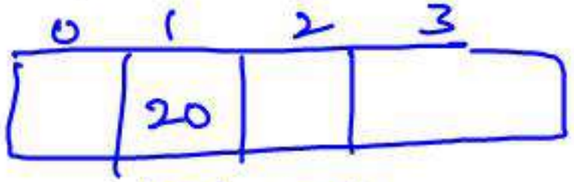
$f = 0, r = 2$

fdequeue()



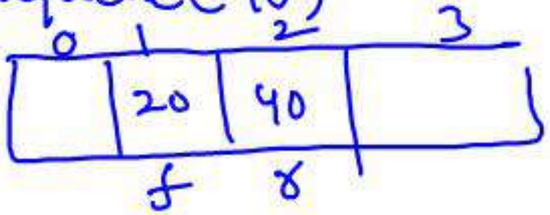
$f = 1, r = 2$

rdequeue();



$f, r = = 1$

enqueue(40) $f = 1, r = 2$



rdequeue $f = 1, r = 1$

rdequeue(); f=1, r=0

rdequeue(); underflow
fdequeue(); underflow

} f>r
f=r=-1

enqueue(5); f=0 r=0

f=0 r=1 enqueue(100)

f=0 r=2 enqueue(15);

f=0 r=3 enqueue(20);

f=0 r=2 rdequeue();

f=0 r=3 enqueue(45);

enqueue(65); f=0 r=4

fdequeue() f=1, r=4

fdequeue() f=2, r=4

rdequeue() f=2, r=3

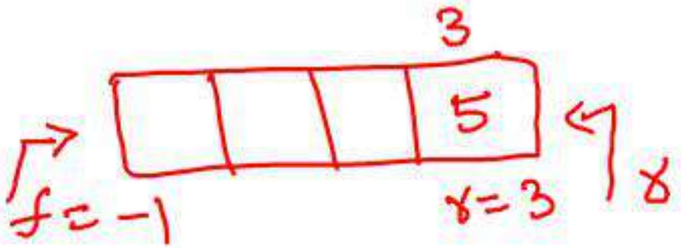
fdequeue() f=3, r=3

size = 4 f = -1 r = 4

Output restricted dequeue()

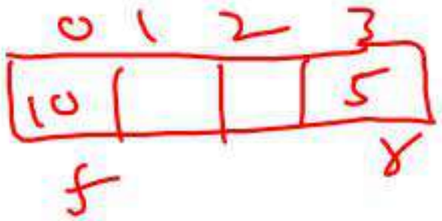
Output from only rear end, input from both ends.

enqueue(5);



f = -1 r = 3

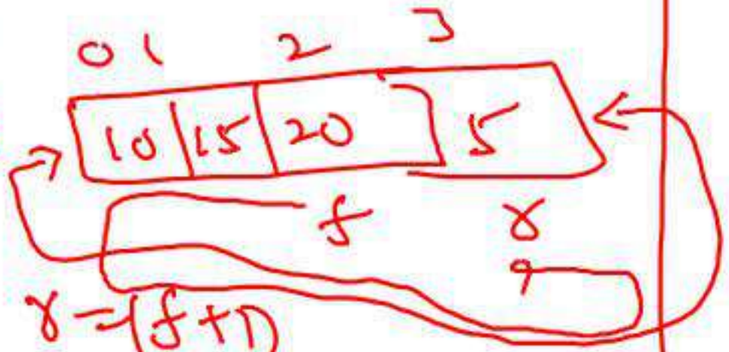
enqueue(10);



f = 0 r = 3

enqueue(15); f = -1, r = 3

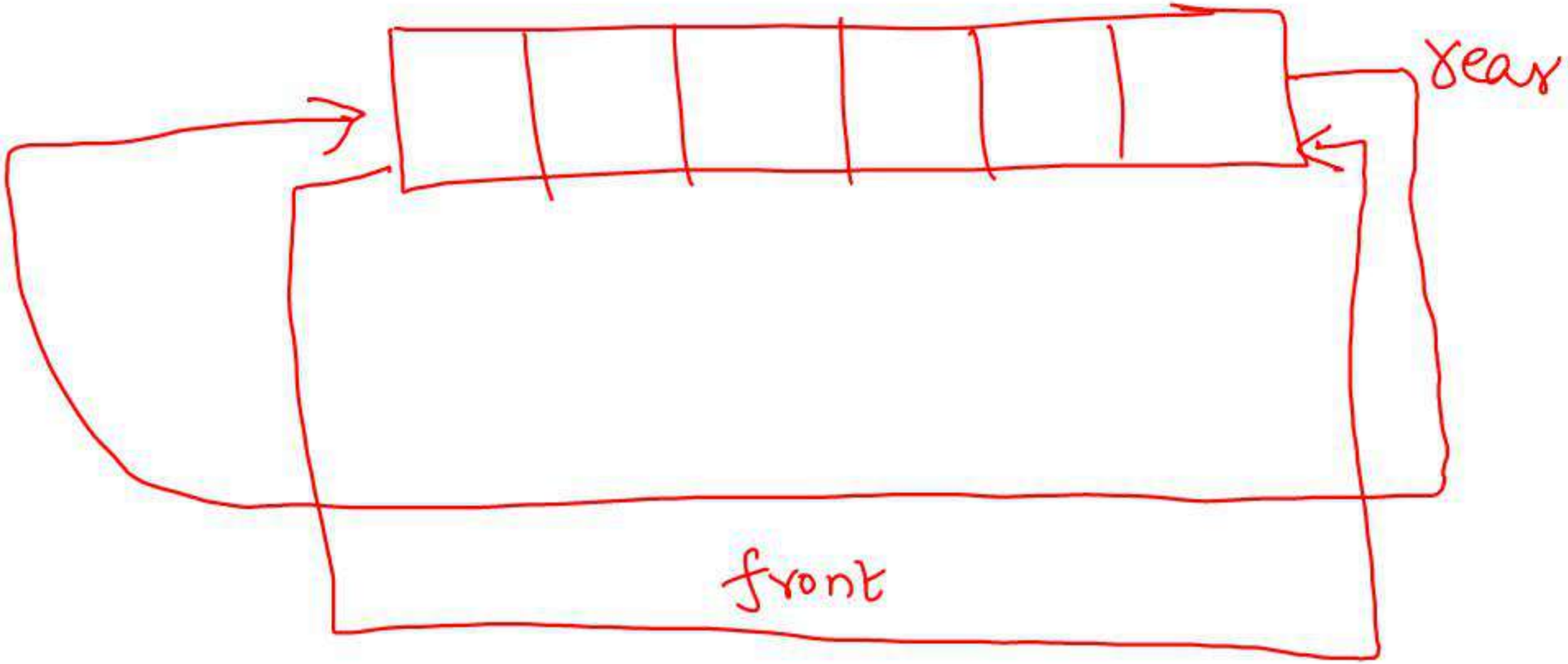
enqueue(20); f = 2, r = 3



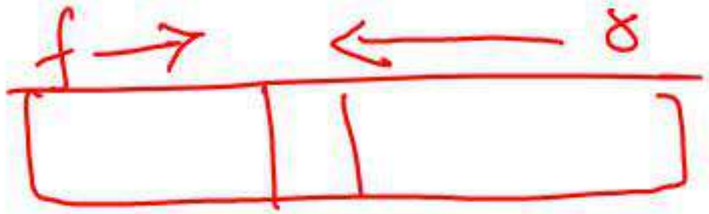
enqueue(25); overflow; r = (f+1)

dequeue();
f = 2, r = 2

Circular array



Array



Insert At Front

(Circular array)

This operation adds an element at the front.

- Check the position of front.
- If front < 1, reinitialize front = n-1 (last index).
- Else, decrease front by 1. $f = f - 1$
- Add the new key into array[front].
- Insert element at the position of front

BYJU'S EXAM PREP

Insert at the Rear

This operation adds an element to the rear.

- Check if deque is full
- If the deque is full, reinitialize rear = 0.
- Else, increase rear by 1. $r = r + 1$
- if deque is not full increase rear
- Add the new key into array[rear].

BYJU'S EXAM PREP

Delete From front

The operation deletes an element from the front.

- Check if the deque is empty.
- If the deque is empty (i.e. front = -1), deletion cannot be performed (underflow condition).
- If the deque has only one element (i.e. front = rear), set front = -1 and rear = -1.
- Else if front is at the end (i.e. front = n - 1), set go to the front, front = 0.
- Else, front = front + 1.

BYJU'S EXAM PREP

Delete From Rear

This operation deletes an element from the rear.

- Check if the deque is empty.
- If the deque is empty (i.e. front = -1), deletion cannot be performed (underflow condition).
- If the deque has only one element (i.e. front = rear), set front = -1 and rear = -1, else follow the steps below.
- If rear is at the front (i.e. rear = 0), set go to the front rear = n - 1.
- Else, rear = rear - 1.

BYJU'S EXAM PREP

Applications Of Deque

- In undo operations on software.
- To store history in browsers.
- For implementing both stacks and queues.

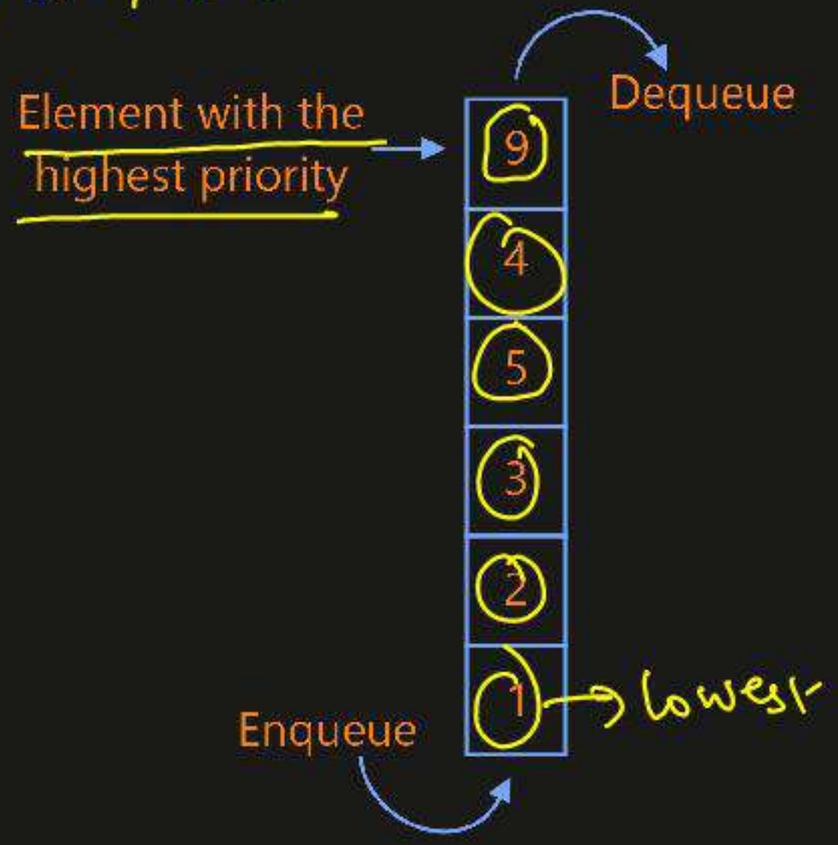
BYJU'S EXAM PREP

Priority Queue

	f=0	f=1	f=2	f=3	f=4
value:	10	20	30	40	50
Priority:	2	5	1	3	4

o/p sequence: 30, 10, 40, 50, 20

- A priority queue is a special type of queue in which each element is associated with a priority value. And elements are served on the basis of their priority. That is, higher priority elements are served first.
- However, if elements with the same priority occur, they are served according to their order in the queue.
- Generally, the value of the element itself is considered for assigning the priority.
- For example, The element with the highest value is considered the highest priority element. However, in other cases, we can assume the element with the lowest value as the highest priority element.
- We can also set priorities according to our needs.



Priority Queue

BYJU'S EXAM PREP

Implementation

- Priority queue can be implemented using an array, a linked list, a heap data structure, or a binary search tree. Among these data structures, heap data structure provides an efficient implementation of priority queues. [Binary - Heap].

Queues - 2

[Circular array] : Random Access

Q1. A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue) ?

[GATE 2016]

A. Both operations can be performed in O(1) time

B. At most one operation can be performed in O(1) time but the worst- case time for the other operation will be Ω(n)

C. The worst- case time complexity for both operations will be Ω(n)

D. Worst case time complexity for both operations will be Ω(log n)

Queues - 2

Q2. Consider the following statements:

i. First-in-first out types of computations are efficiently supported by STACKS.

ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations. (Dynamic memory allocation)

iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices. (No Memory wastage)

iv. Last-in-first-out type of computations are efficiently supported by QUEUES.

Which of the following is correct? [GATE 1996]

A. (ii) and (iii) are true

B. (i) and (ii) are true

C. (iii) and (iv) are true

D. (ii) and (iv) are true

BYJU'S EXAM PREP

Q3. Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operations are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are –

[GATE 2012]

A. Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT

B. Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR

C. Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT

D. Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

Queues - 2

Home work

Q4. Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are: [GATE 2007]

i. isEmpty (Q) — returns true if the queue is empty, false otherwise.

ii. delete (Q) — deletes the element at the front of the queue and returns its value.

iii. insert (Q, i) — inserts the integer i at the rear of the queue.

What operation is performed by the above function f ?

Consider the following function:

```
void f (queue Q) {
  int i ;
  if (!isEmpty(Q)) {
      i = delete(Q);
      f(Q);
      insert(Q, i);
  }
}
```

A. Leaves the queue Q unchanged

B. Reverses the order of the elements in the queue Q

C. Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

D. Empties the queue Q

Queues - 2

BYJU'S EXAM PREP

Home work

Q5. A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.

Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure ?

[GATE 2018]

A. $\Theta(1), \Theta(1)$

B. $\Theta(1), \Theta(n)$

C. $\Theta(n), \Theta(1)$

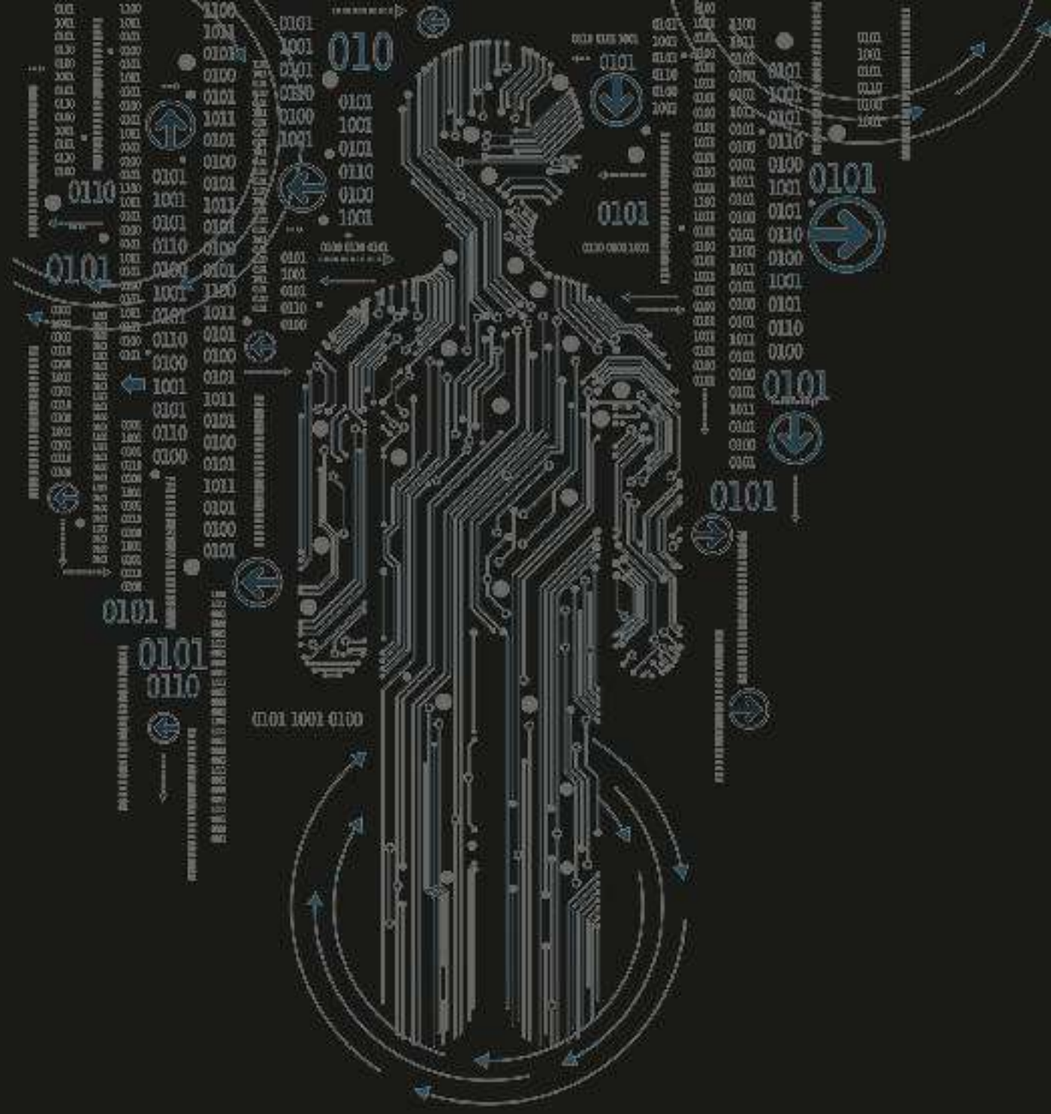
D. $\Theta(n), \Theta(n)$

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Stack and Queues Together



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Queues — FIFO linear DS
- enqueue
- Dequeue

} different Ends

- Simple Queue
- Circular Queue
- Double Ended Queue
- Priority Queue

} Queues – 2

Implementation
- Array ✓
- Stack
- Linked List ✓

BYJU'S EXAM PREP

- Stack Using 2 queues
- Stack Using 1 queue

Stack Implementation Using 2 Queues

A stack is a Last–In, First–Out (LIFO) data structure in which elements are inserted and removed from one end of the stack known as the top of the stack. It has two major operations: The push operation, which adds an element into the stack, and the pop operation, which removes the most recently added element from the stack, but not yet removed.

There are several ways to implement a stack using one or two queues by tweaking their enqueue and dequeue operations.

Stack and Queues Together

Push Operation Costly

Push : O(n) Linear time

Pop : O(1) Constant time

The idea is to implement the queue's enqueue operation such that the last entered item always ends up at the queue's front. To achieve this, we need an additional queue.

- To push an item into the stack, first move all elements from the first queue to the second queue, then enqueue the new item into the first queue, and finally move all elements back to the first queue. This ensures that the new item lies in front of the queue and hence would be the first one to be removed.
- To pop an item from the stack, return the front item from the first queue. dequeue(Q_1)

Stack and Queues Together

Actual o/p Sequence : 30, 40, 50, 20

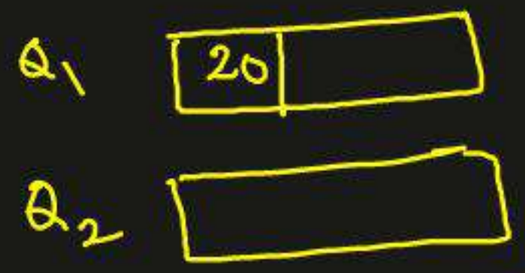
Expected o/p Sequence : 30, 40, 50, 20

Example – push O(n), pop O(1)

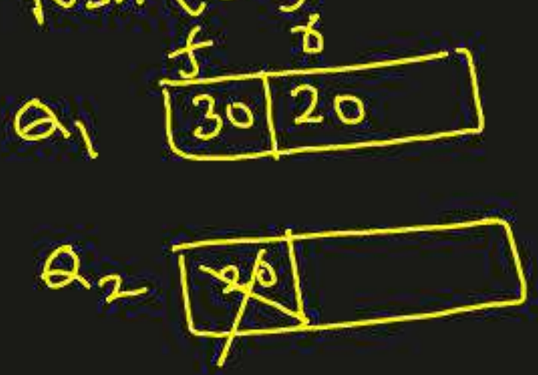
Let Operations to be performed On stack are :

- Push(20)
- Push(30)
- Pop
- Push(40)
- Pop
- Push(50)
- Pop
- Pop

Push (20)



Push (30)

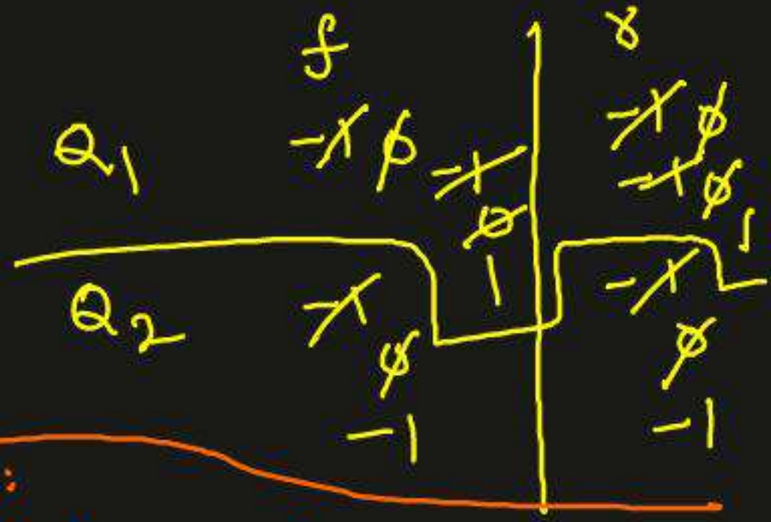


Pop( )

o/p. 30

f = 1, r = 1

Let there are 2 Queues, initially Empty [Q1, Q2]



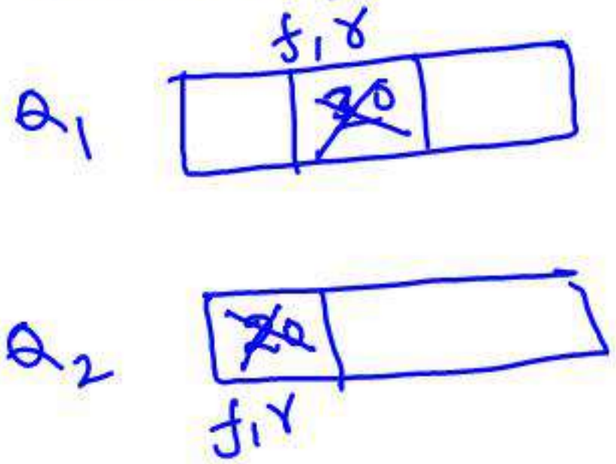
PUSH:

[Q1 → Q2] (enqueue Q1) [Q2 → Q1]

n + 1 + n

⇒ 2n+1 ⇒ O(n)

Push(40)



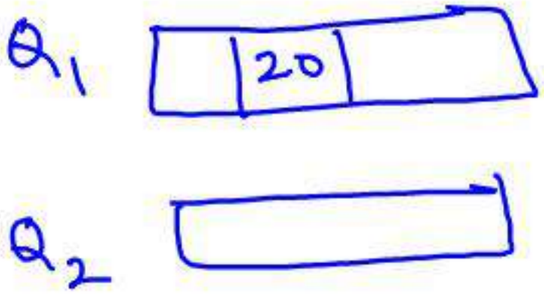
$f=2, r=1$

$f>r$: underflow

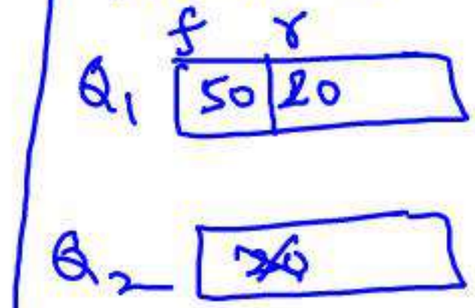
$f=r=-1$

	f	r
Q_1	$\cancel{1}\ \cancel{2}\ \cancel{-1}\ \cancel{0}_{1}$	$\cancel{1}\ \cancel{-1}\ \cancel{0}_{1}$
Q_2	$\cancel{-1}\ \cancel{0}_{-1}$	$\cancel{-1}\ \cancel{0}_{-1}$

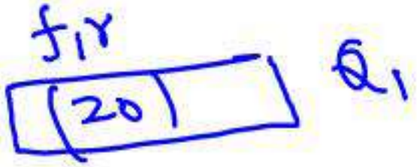
Pop ( ) : dequeue(Q_1) : o/p: 40



Push(50)



Pop( )



dequeue(q_1) : o/p : 50

Pop ( ) Q_1 f, r = -1

dequeue(q_1) : o/p : 20

Stack and Queues Together

Pop Operation Costly

Push O(1) Constant time

Pop O(n) linear time

Note that the elements are exchanged between the queue twice for every push operation. This can impact performance if push operations are frequent. Here's an alternative approach that affects the pop operation's time complexity instead of the push operation.

- To push an item into the stack, enqueue the item to the first queue. enqueue(Q_1)
- To pop an item from the stack, move all (n-1) elements from the first queue to the second queue except the last element, and then return the last element after moving all elements back to the first queue.

dequeue(Q_1) (n-1) back to Q_1

Stack and Queues Together

Example – push O(1), pop O(n)

Let Operations to be performed On stack are :

- Push(20)
- Push(30)
- Pop
- Push(40)
- Pop
- Push(50)
- Pop
- Pop

Expected o/p sequence : 30, 40, 50, 20

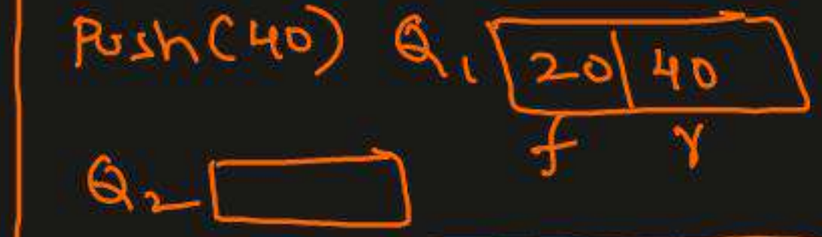
Actual o/p sequence : 30, 40, 50, 20

	f	r
Q_1	~~-1~~ ~~0~~ ~~1~~ ~~-1~~ 0	~~-1~~ ~~0~~ ~~1~~ ~~-1~~ ~~0~~ 1
Q_2	~~-1~~ ~~0~~ -1	~~-1~~ ~~0~~ -1

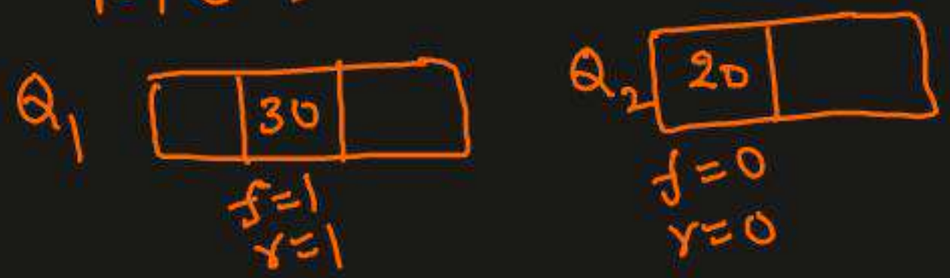
Push(20)

Push(30)

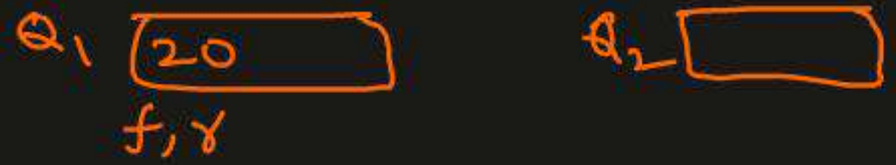
Push(40)



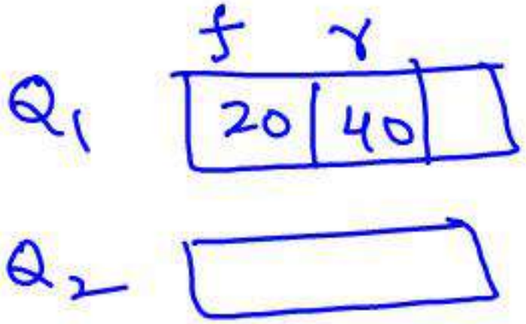
Pop( )



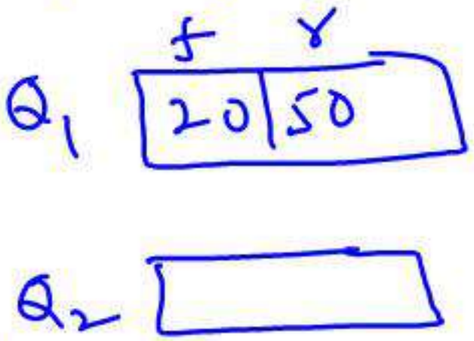
dequeue(Q_1) : o/p : 30



BYJU'S EXAM PREP

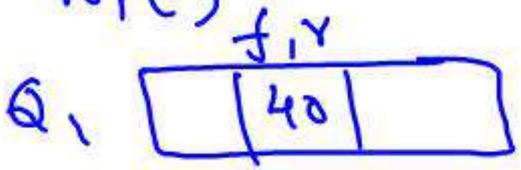


Push(50)



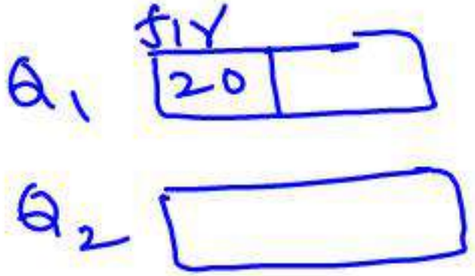
	f	r
Q_1	ϕ ~~x~~ ~~-1~~ ϕ ~~x~~ ~~-1~~ 0	~~x~~ ~~-1~~ ϕ ~~x~~ ~~-1~~ 0
Q_2	~~-1~~ ϕ ~~-1~~ ϕ -1	~~-1~~ ϕ ~~-1~~ ϕ -1

Pop()

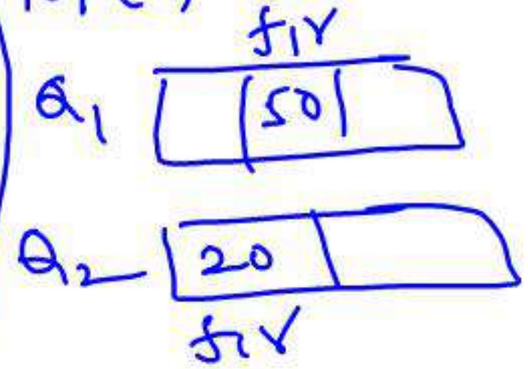


dequeue(Q_1) o/p: 40

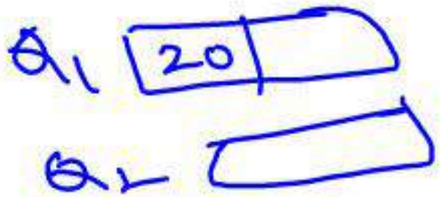
$f = r = -1$



Pop()

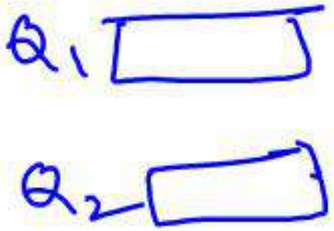


dequeue(Q_1) o/p: 50



Pop()

dequeue(Q_1) o/p: 20



BYJU'S EXAM PREP

Using one queue with call stack

Recursive implementation.

f(x)
f(5) x = 5
↳ f(4) x =
↳ f(3) x =

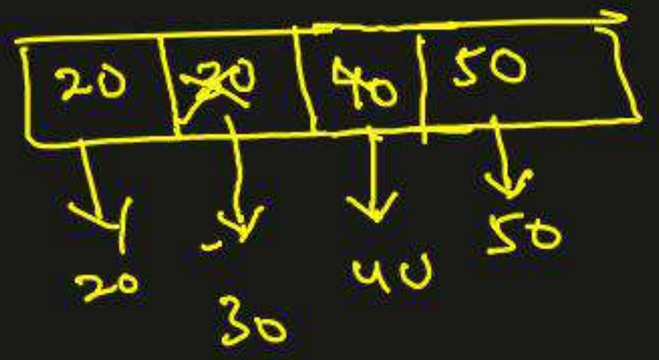
We can also use an implicit stack (call stack) along with a queue to construct a stack.

Let Operations to be performed On stack are :

- Push(20)
- Push(30)
- Pop
- Push(40)
- Pop
- Push(50)
- Pop
- Pop

R_Queue (enqueue (·))

Head Recursion - for Push
→ Printing for Pop



disadvantage

For Every Recursive call, memory is allocated internally
↓
Space complexity is high.

BYJU'S EXAM PREP

simple

Queue Implementation Using Stack

Queue is a linear data structure that follows FIFO (First In First Out) principle in which insertion is performed from the rear end and the deletion is done from the front end. Stack is a linear data structure that follows LIFO (Last In First Out) principle in which both insertion and deletion are performed from the top of the stack.

There are two approaches to implement Queue using Stack : [2 stacks]

- Making a dequeue operation costly
- Making a enqueue operation costly

- Using one recursive stack also. [Space complexity high]

BYJU'S EXAM PREP

Enqueue Costly

enqueue O(n)
dequeue O(1)

If we implement the Queue using Stack by making a enqueue operation costly means that time complexity in enqueue operation would be O(n) and the time complexity in dequeue operation would be O(1).

Enqueue: We will consider two stacks named as stack1 and stack2. In case of enqueue operation, first all the elements will be popped from the stack1 and push it into the stack2. Once all the elements from the stack1 are pushed into the stack2, then the new element is added in the stack1. After adding the new element in the stack1, all the element are moved back from stack1 to stack2. Here, the time complexity of enqueue operation would be O(n).

Dequeue: In stack1, the oldest element would be at the top of the stack, so time taken to perform a dequeue operation would be O(1).

Pop (S1).

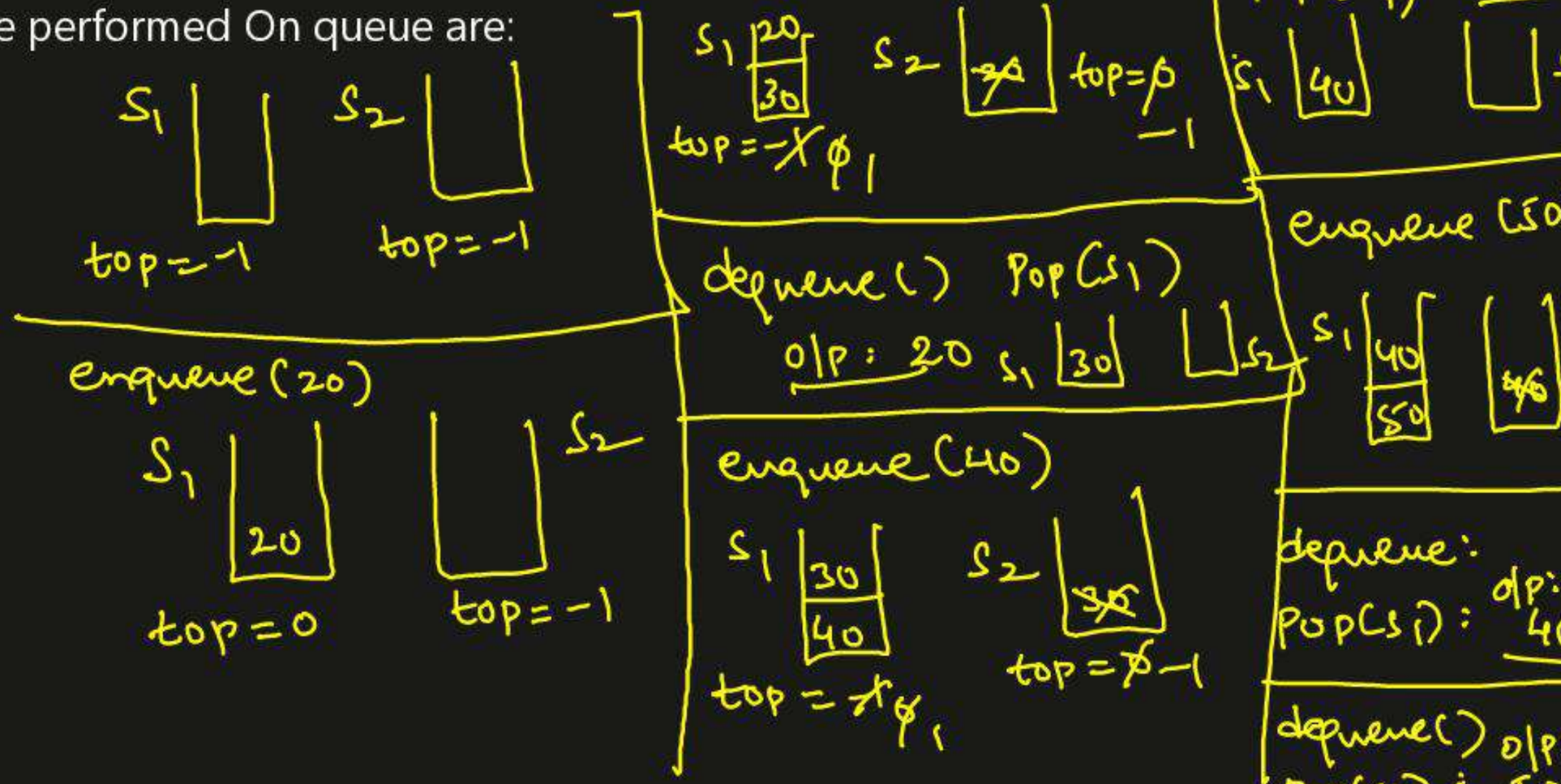
BYJU'S EXAM PREP

Example – enqueue O(n), Dequeue O(1)

Expected o/p Seq : 20, 30, 40, 50
Actual : 20, 30, 40, 50

Let Operations to be performed On queue are:

- enqueue(20)
- enqueue(30)
- dequeue
- enqueue(40)
- dequeue
- enqueue(50)
- dequeue
- dequeue



BYJU'S EXAM PREP

Dequeue Costly

If we implement the Queue using Stack by making a dequeue operation costly means that time complexity in enqueue operation would be O(1) and the time complexity in dequeue operation would be O(n).

Enqueue: when we insert the elements in the stack then the elements are added on the top of the stack so it takes O(1) time. push(x)

Dequeue : We need to consider two stacks named as Stack1 and Stack2. First, we insert the elements in the Stack1 and then we remove all the elements from the Stack1. Once all the elements are popped from the Stack1 then they are added in the Stack2. The topmost element would be popped out from the Stack2 and then all the elements from the Stack2 are moved back to Stack1. Here, dequeue operation is performed two times on the data so time complexity is O(n).

BYJU'S EXAM PREP

Example – enqueue O(1), Dequeue O(n)

enqueue( ) : Push(S_1)

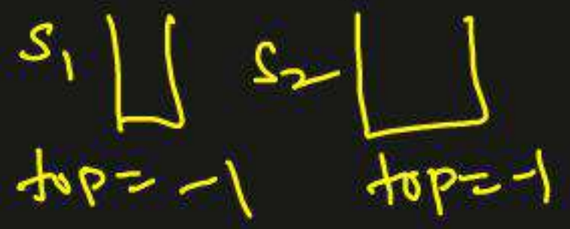
dequeue( ) : 1) (n-1) $S_1 \rightarrow S_2$

2) Pop(S_1)

3) (n-1) $S_2 \rightarrow S_1$

Let Operations to be performed On queue are:

- enqueue(20)
- enqueue(30)
- dequeue
- enqueue(40)
- dequeue
- enqueue(50)
- dequeue
- dequeue



Practice on self

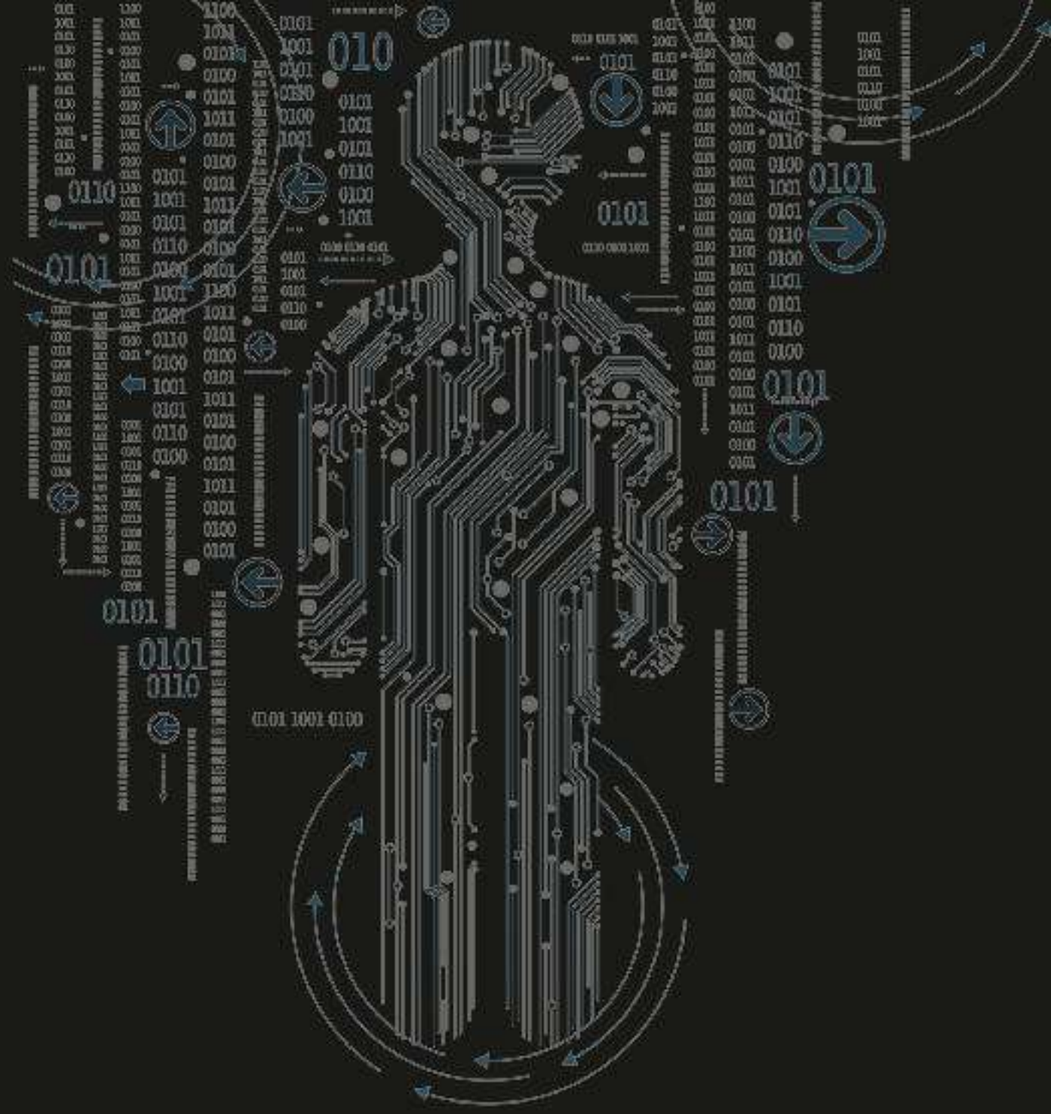
Tomorrow @ 10 AM

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Fundamentals of Binary Tree



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Arrays, Linked Lists, Stack & Queue

Linear Data Structures

Expression Evaluation

Prefix Expression

1) Scan Exp R to L
2) If operand : Push

   Operator : Pop first 2 operands, Perform operation, Push back result.

a op b

Postfix Expression

1) Scan Exp L to R
2) If operand : Push

   Operator : - Pop first 2 operands
   - Perform operation
   - Push back result.

b op a

Fundamentals of Binary Tree

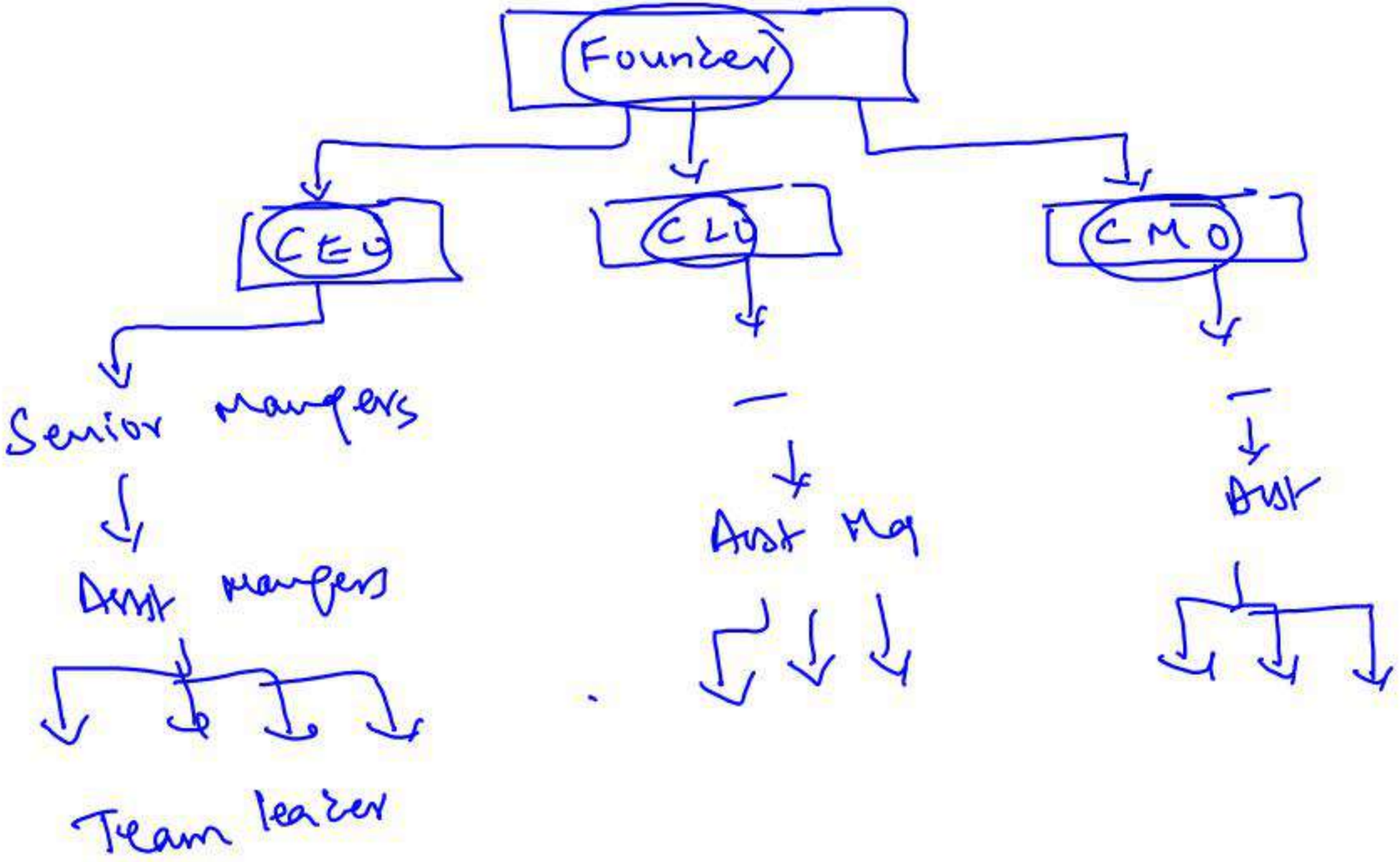
BYJU'S EXAM PREP

Tree (Hierarchical Data structure)

or elements or nodes

- A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
- A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels. [Random memory locations]
- In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type. In the above tree structure, the node contains the name of the employee, so the type of data would be a string.
- Each node contains some data and the link or reference of other nodes that can be called children.

Amazon Employee hierarchy.



BYJU'S EXAM PREP

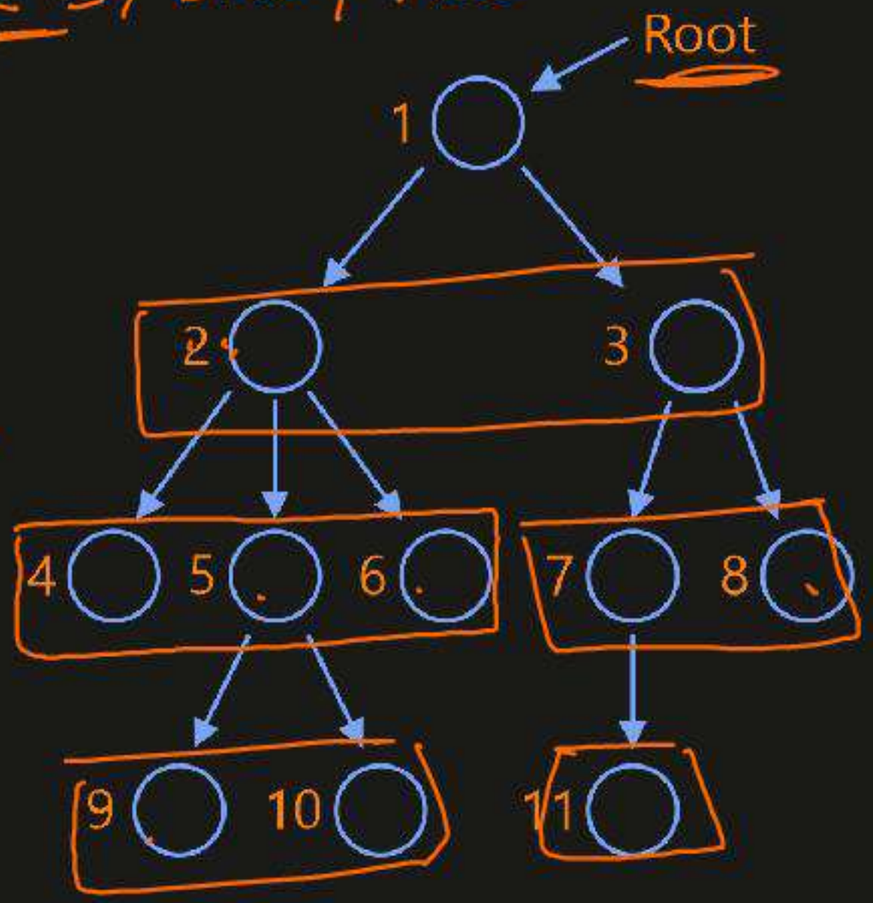
Tree – Basic Terms

Max No. of children per Node == "n" => n-ary tree

n == 2 => Binary Tree

In the above structure, each node is labeled with some number. Each arrow shown in the above figure is known as a link between the two nodes.

- Root: The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. In the above structure, node numbered 1 is the root node of the tree. If a node is directly linked to some other node, it would be called a parent-child relationship.
- Child node: If the node is a descendant of any node, then the node is known as a child node.
- Parent: If the node contains any sub-node, then that node is said to be the parent of that sub-node.
- Sibling: The nodes that have the same parent are known as siblings.



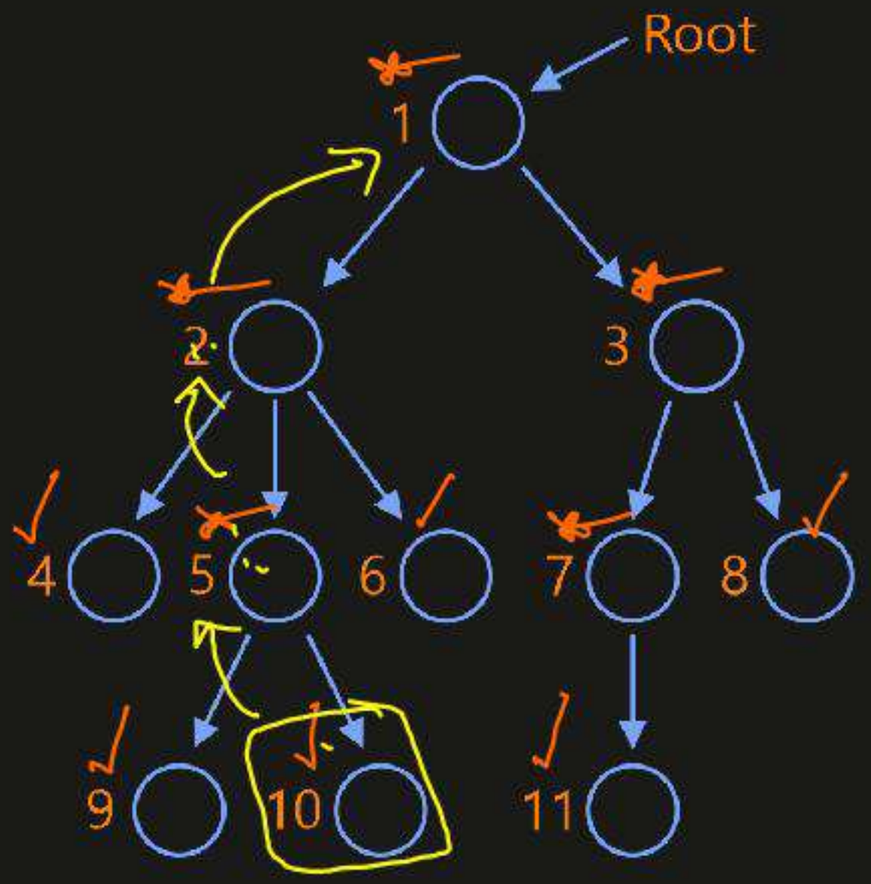
Tree – Basic Terms

Ternary Tree

BYJU'S EXAM PREP

Tree – Basic Terms

- Leaf Node: The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.
- Internal nodes: A node has atleast one child node known as an internal (Non-leaf Nodes)
- Ancestor node: An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, nodes 1, 2, and 5 are the ancestors of node 10.
- Descendant: The immediate successor of the given node is known as a descendant of a node. In the above figure, 10 is the descendant of node 5.

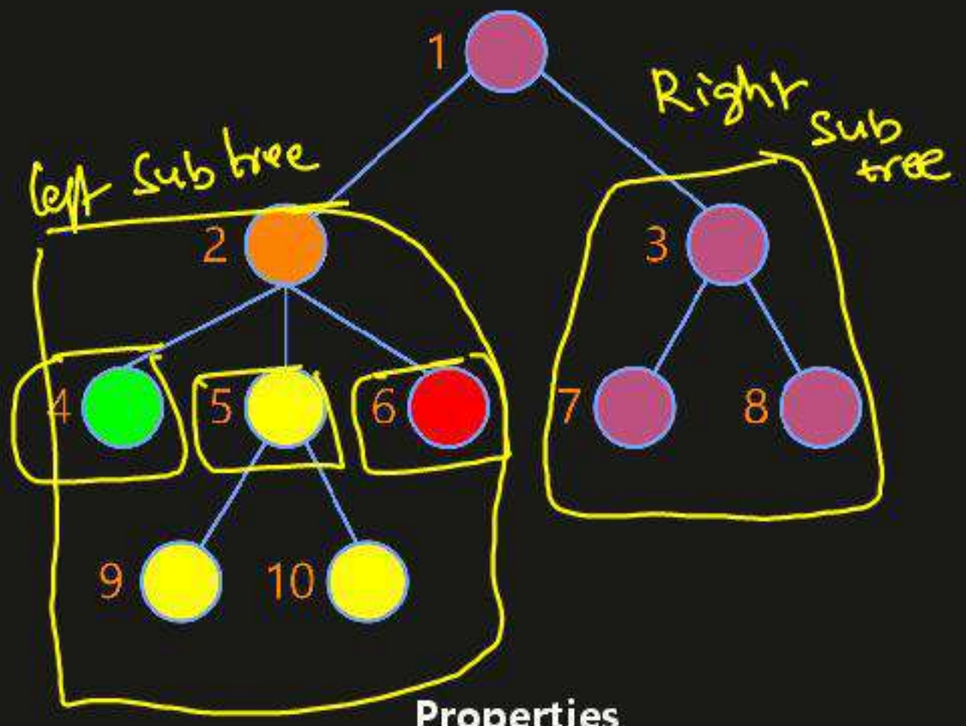


Tree – Basic Terms

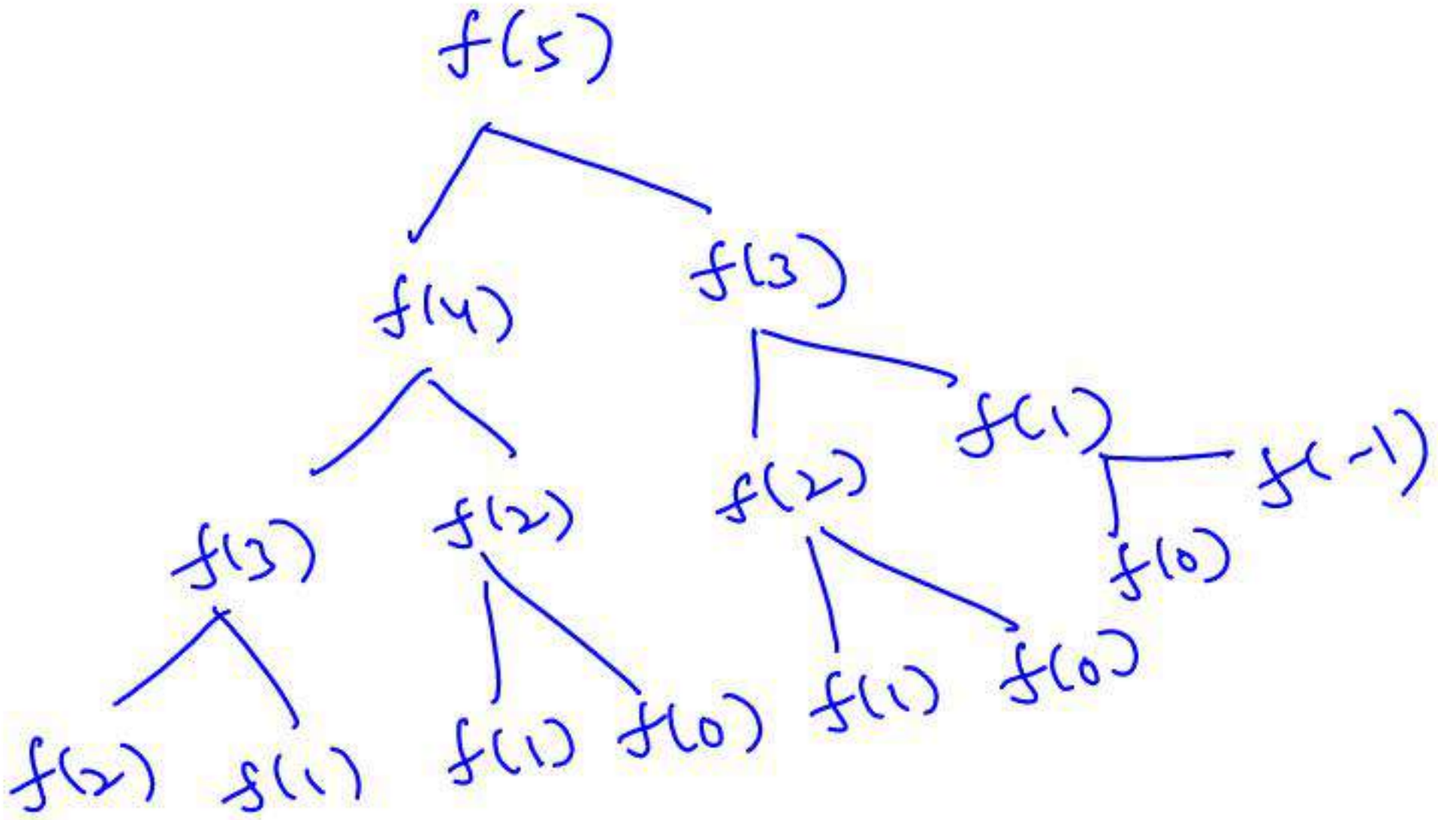
BYJU'S EXAM PREP

Properties

- Recursive data structure: The tree is also known as a recursive data structure.
- A tree can be defined as recursively because the distinguished node in a tree data structure is known as a root node.
- The root node of the tree contains a link to all the roots of its subtrees.
- The left subtree is shown in the yellow color in the below figure, and the right subtree is shown in the red color.
- The left subtree can be further split into subtrees shown in three different colors.
- Recursion means reducing something in a self-similar manner. So, this recursive property of the tree data structure is implemented in various applications.



Properties

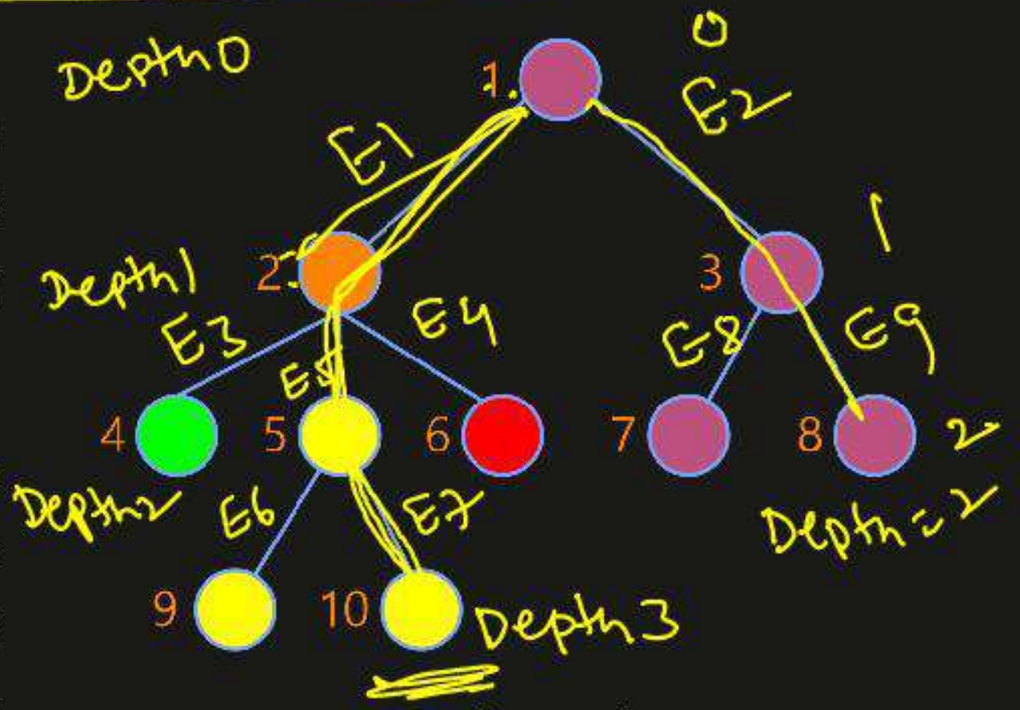

f(5)
f(4)
f(3)
f(3)
f(2)
f(2)
f(1)
f(-1)
f(0)
f(2)
f(1)
f(1)
f(0)
f(1)
f(0)

BYJU'S EXAM PREP

Properties

Level of Tree == Depth of tree == height of tree

- Number of edges: If there are n nodes, then there would n-1 edges. Each arrow in the structure represents the link or path. Each node, except the root node, will have atleast one incoming link known as an edge. There would be one link for the parent-child relationship.
- Depth of node x: The depth of node x can be defined as the length of the path from the root to the node x. One edge contributes one-unit length in the path. So, the depth of node x can also be defined as the number of edges between the root node and the node x. The root node has 0 depth.
- Height of node x: The height of node x can be defined as the longest path from the node x to the ~~leaf~~ root node.



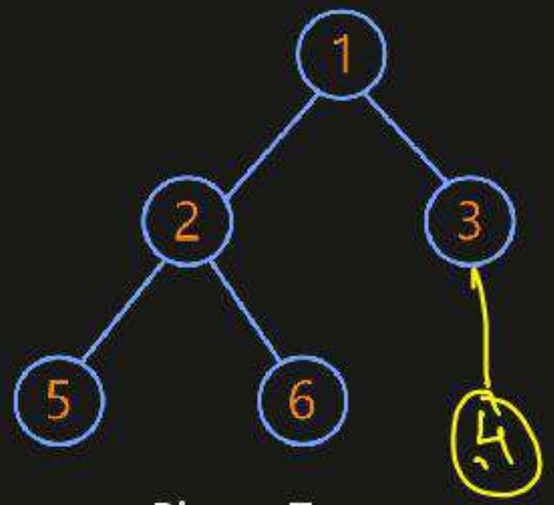
Properties

BYJU'S EXAM PREP

Binary Tree

Number of children of node 'x' == Degree of node 'x'

The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.



Binary Tree

Node	Degree
1	2
2	2
3	1
4	0
5	0
6	0

BYJU'S EXAM PREP

{0, 1, 2}

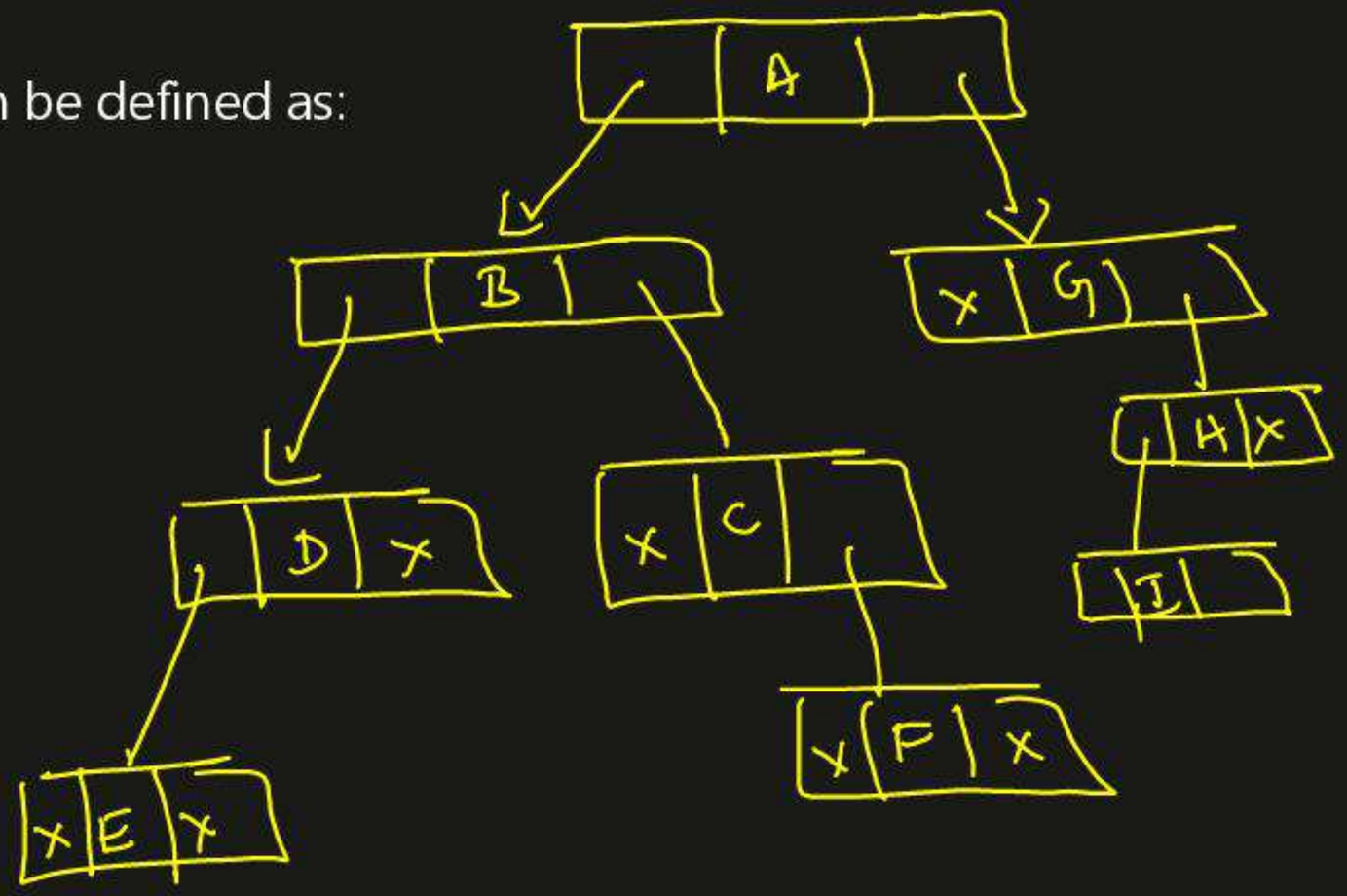
Structure of a Node

(A) : Binary Tree

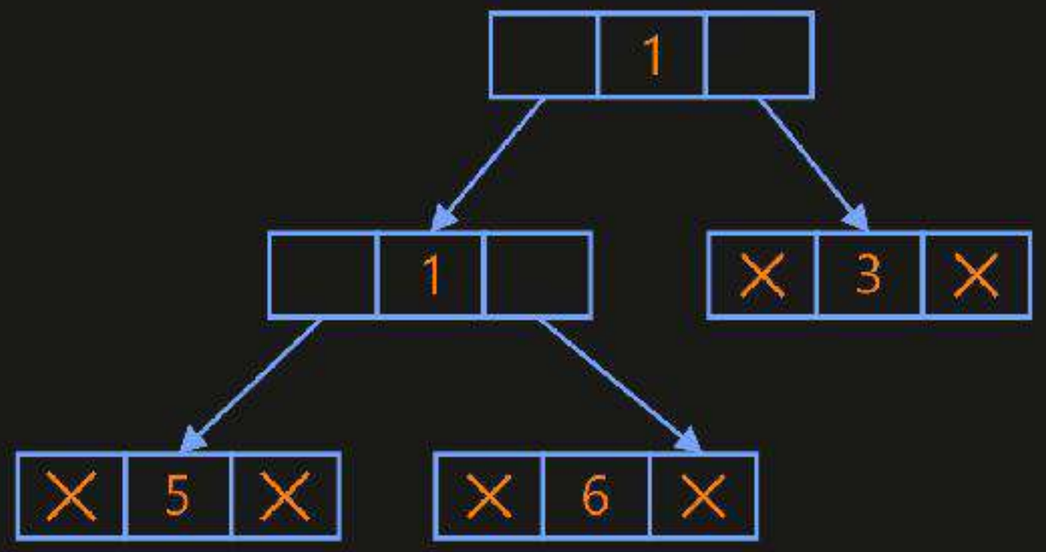
In programming, the structure of a node can be defined as:

```
struct node
{
  int data;
struct node *left;
struct node *right;
}
```

DLL



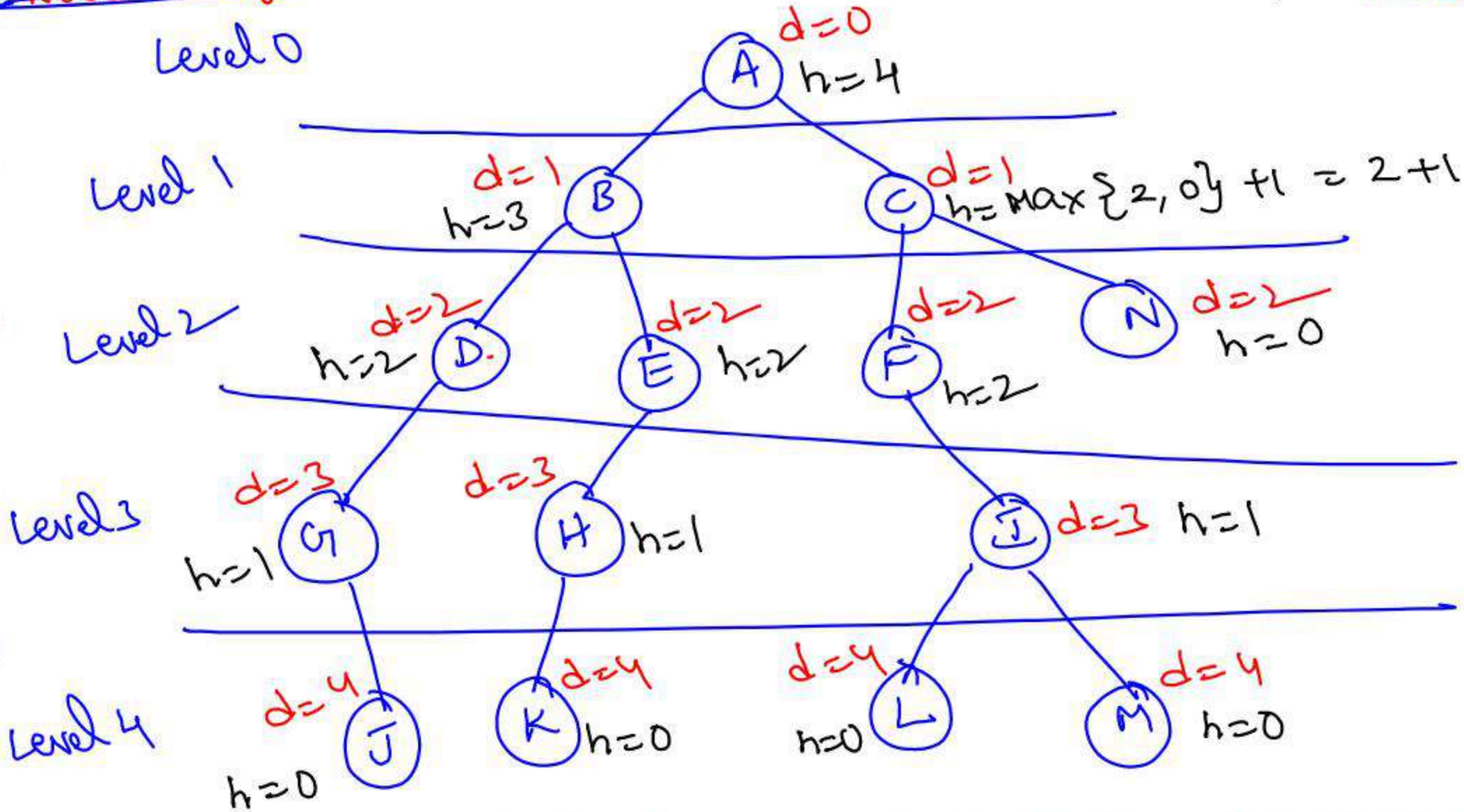
DS Representation



DS Representation

Level of Node == Depth of Node

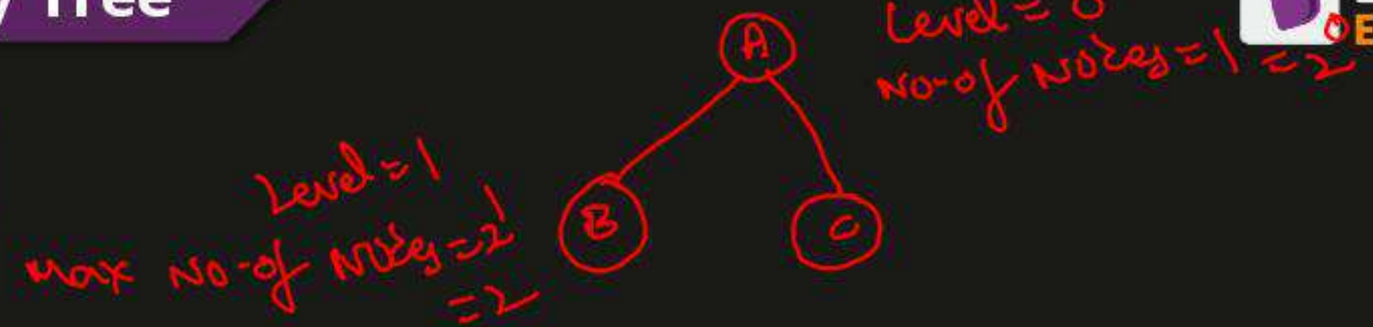
height of Node = Max{children heights} +1



4 == Level of Tree == Depth of Tree = height of Tree

BYJU'S EXAM PREP

Properties of Binary Tree



- At each level of i, the maximum number of nodes is 2^i.
- The height of the tree is defined as the longest path from the root node to the leaf node.
- If the number of nodes is minimum, then the height of the tree would be maximum. (Skewed Trees)
- Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.
- With 'n' Nodes:

  Minimum Height: $h = \log_2(n+1) - 1$

  Maximum Height: $h = n-1$

BYJU'S EXAM PREP

Types Of Binary Tree

Different Types Of Binary Tree include:

- Full /Proper / Strict Binary Tree
- Complete Binary Tree
- Perfect Binary Tree
- Skewed Binary Tree
- Binary Search Tree
- Binary Heap
- AVL Tree
- Red – Black Tree
- Splay Tree etc.

BYJU'S EXAM PREP

Full Binary Tree

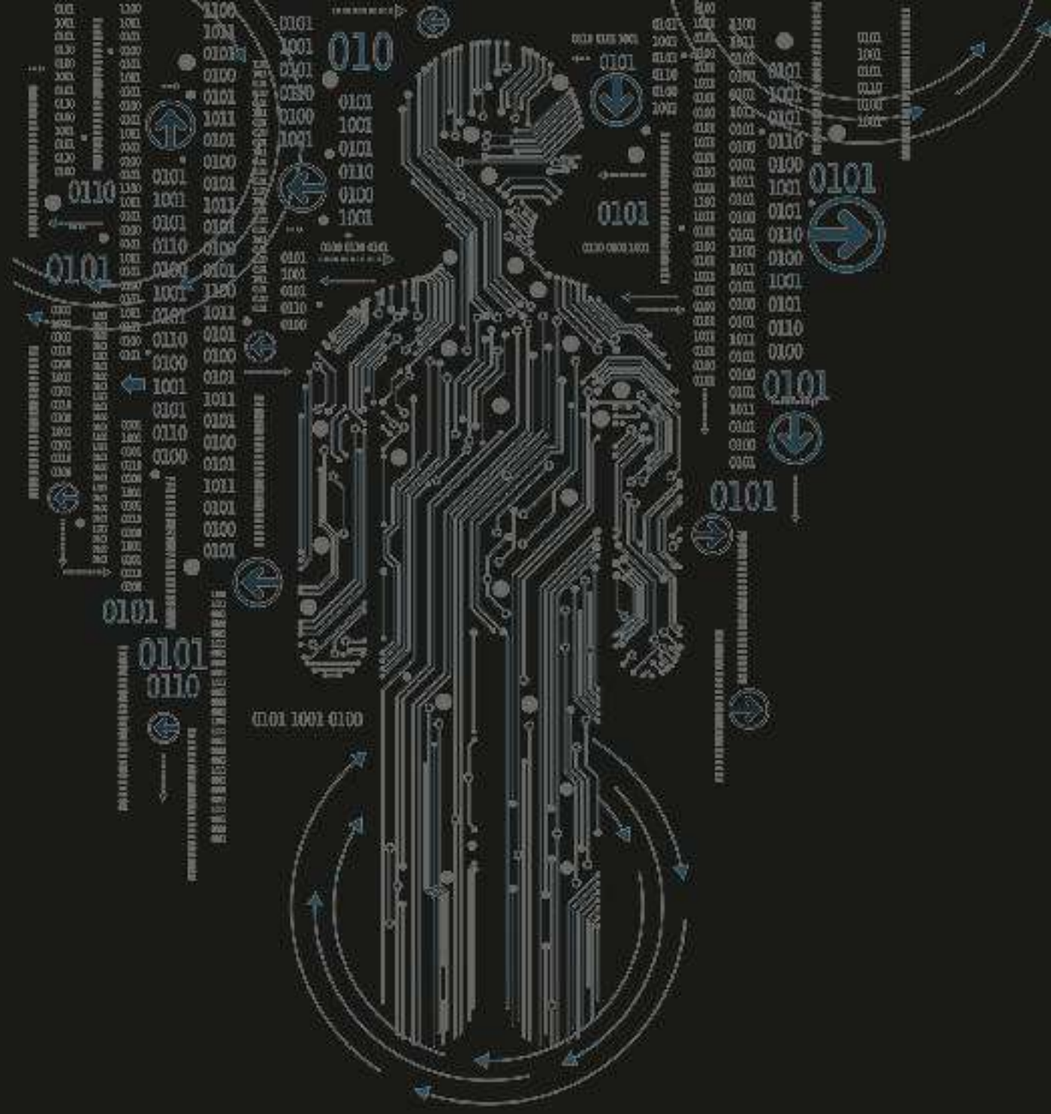
To be Contd...

The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Binary Search Tree



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Types Of Binary Tree

Different Types Of Binary Tree include:

- Full /Proper / Strict Binary Tree
- Complete Binary Tree
- Perfect Binary Tree
- Skewed Binary Tree
- Binary Search Tree
- Binary Heap
- AVL Tree
- Red – Black Tree
- Splay Tree etc.

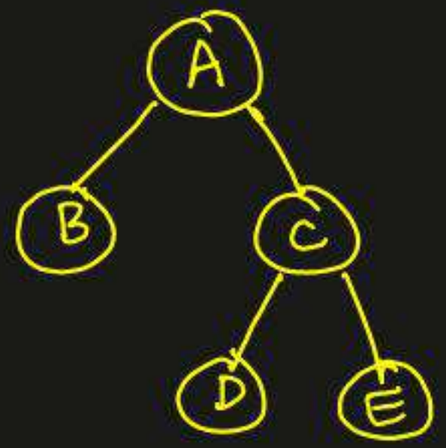
BYJU'S EXAM PREP

Full Binary Tree

The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.

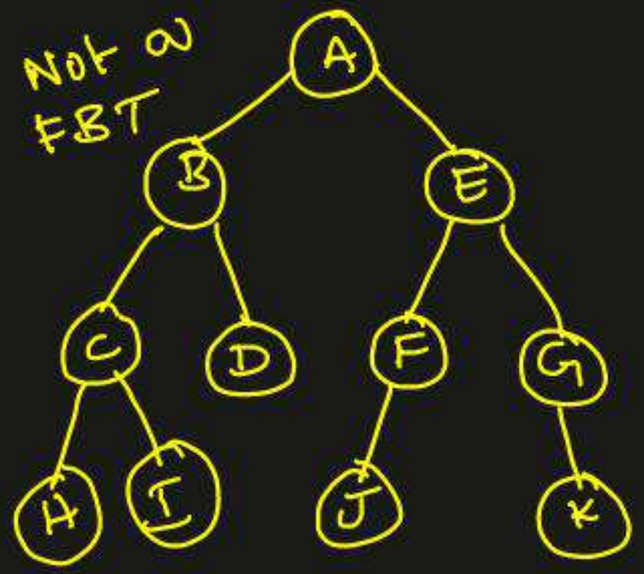
Ex: 1

FBT



Node	A	B	C	D	E
child	2	0	2	0	0

Not a FBT



Nodes F, G has 1 child

Ex: 3

No. of children for A = 0

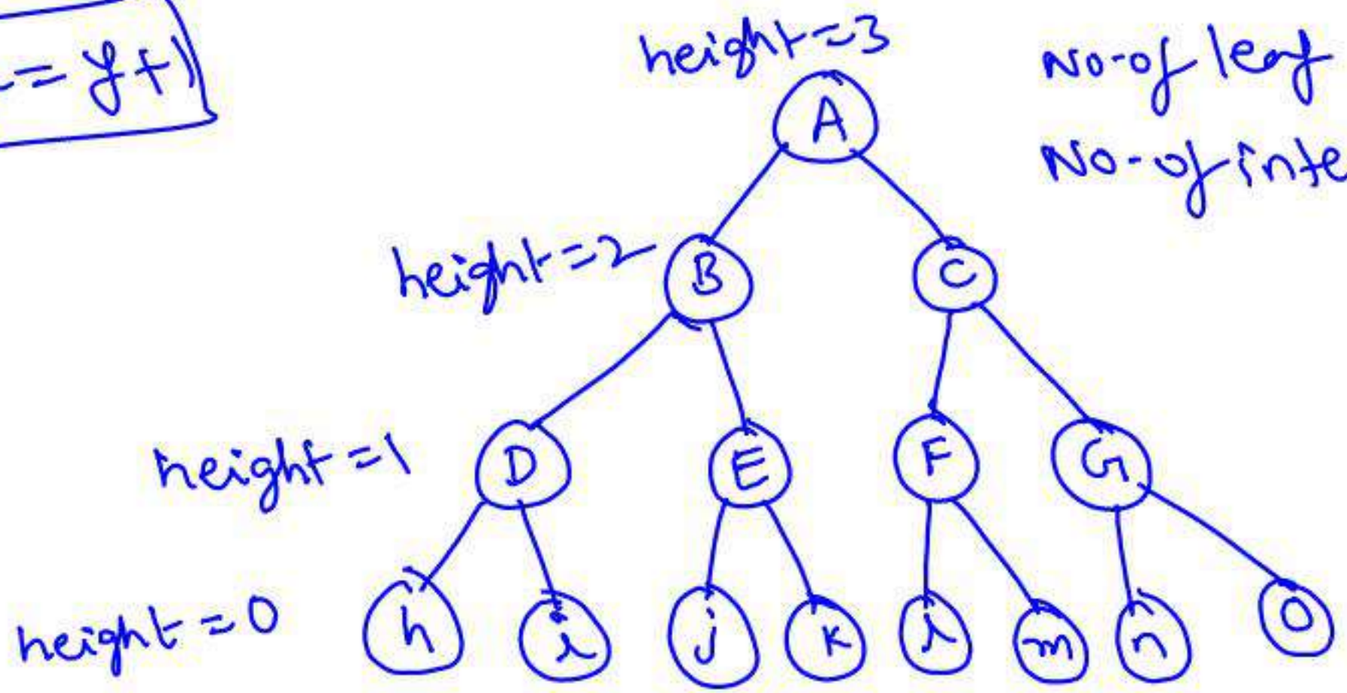
A, B, C has 1 child each

Not a FBT

Full Binary Tree - Properties

- The number of leaf nodes is equal to the number of internal nodes plus 1.
- The maximum number of nodes is the same as the number of nodes in the binary tree, i.e., $2^{h+1} - 1$.
- The minimum number of nodes in the full binary tree is $2*h+1$. $(2*h+1)$
- The minimum height of the full binary tree is $\log_2^{(n+1)} - 1$.
- The maximum height of the full binary tree is $n+1/2$ $(n-1)/2$

$\boxed{x = y+1}$



No-of leaf Nodes $= 8 = x$

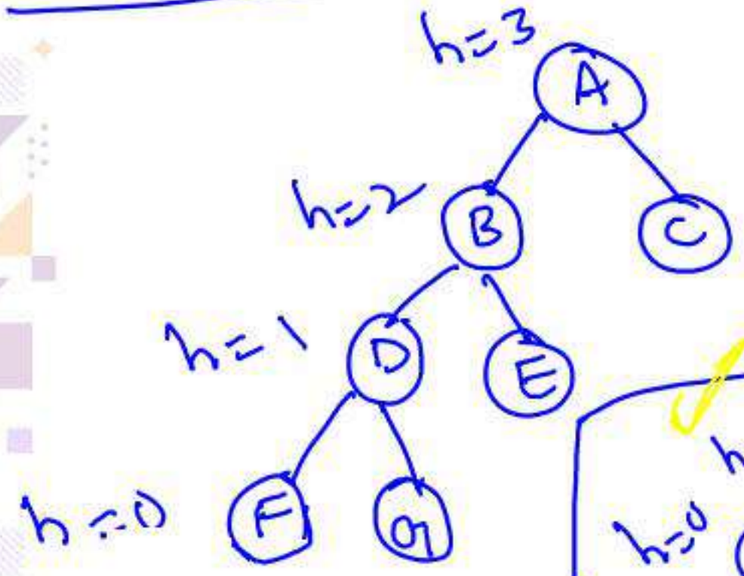
No-of internal Nodes $= 7 = y$

Total No-of Nodes $= 15$

Max No-of Nodes

$= 2^{(h+1)} - 1$

$= 2^{(3+1)} - 1 = 2^4 - 1 = \underline{\underline{15}}$

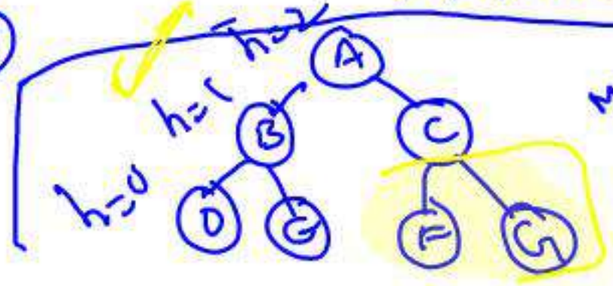


Total No-of Nodes $= 7$

Minimum No-of Nodes $= (2 * h) + 1$

$= (2 * 3) + 1$

$= 7$

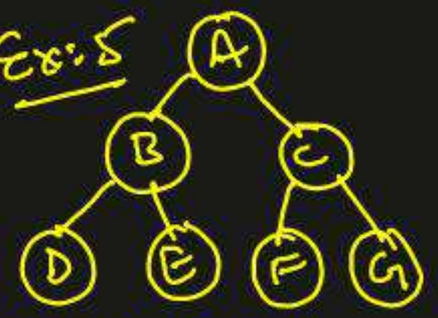


min height $\log_2 z+1 - 1$

$= \log_2 8 - 1 = 3 - 1 = 2$

Complete Binary Tree

Ex: 5



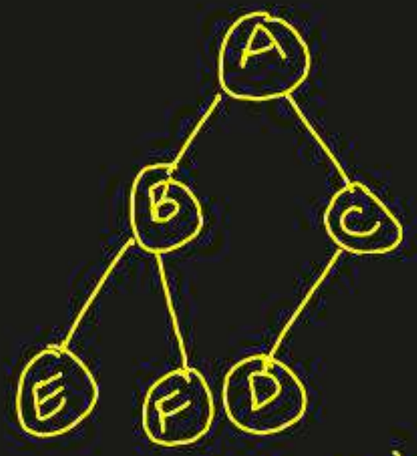
CBT, FBT

Every CBT is Not FBT,

Every FBT is Not CBT.

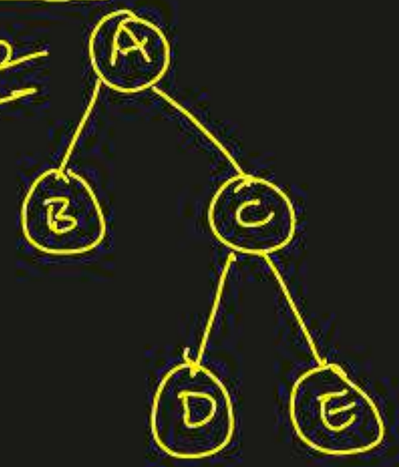
The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.

Ex: 1



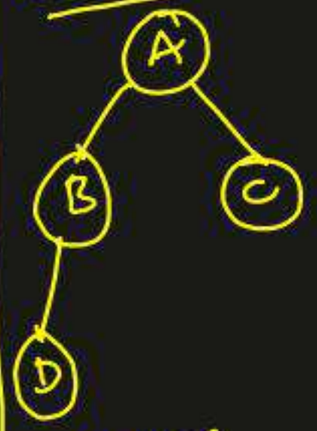
CBT but not FBT

Ex: 2



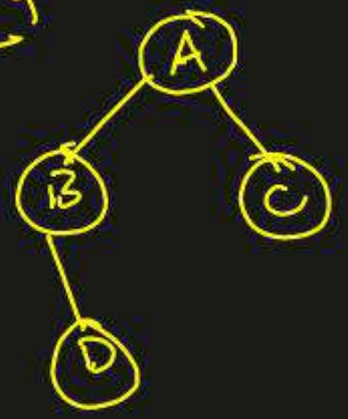
Not CBT, but, FBT

Ex: 3



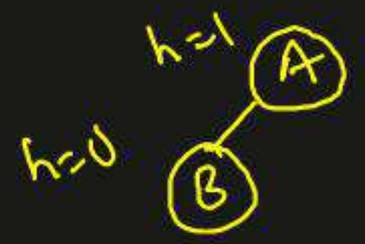
CBT, but not FBT

Ex: 4



Not CBT, Not FBT (left first, Then right)

BYJU'S EXAM PREP



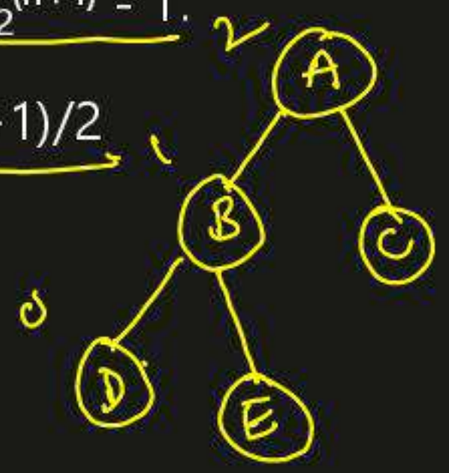
$2^1 = 2$ nodes (min)

CBT Properties

- The maximum number of nodes in complete binary tree is $2^{h+1} - 1$.
- The minimum number of nodes in complete binary tree is 2^h.
- The minimum height of a complete binary tree is $\log_2^{(n+1)} - 1$.
- The maximum height of a complete binary tree is $(n+1)/2$

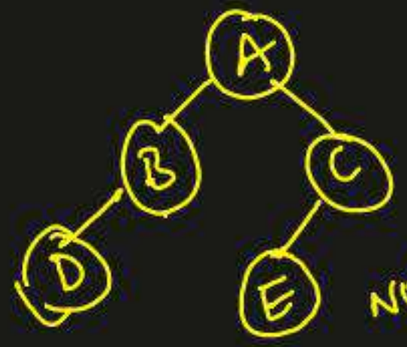
Ex: 5

$2^{2+1} - 1 = 2^3 - 1 = 7$



$n = 5$

$\frac{5+1}{2} = \frac{6}{2} = 3$



Not CBT

BYJU'S EXAM PREP

n nodes — Maximum height $(n-1)$ — Skewed Binary Tree

minimum height $\left[\log_2^{n+1} - 1\right]$ — Perfect Binary Tree (FBT, CBT)

---

height : h — Minimum nodes $= (h+1)$ — Skewed Binary Tree

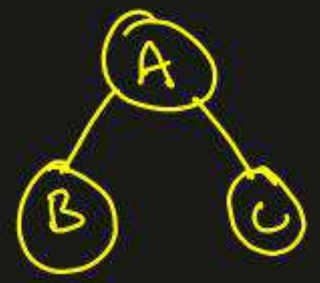
Maximum nodes $= (2^{h+1} - 1)$ — Perfect Binary Tree

Every PBT is FBT, Every PBT is CBT but not vice versa | Not vice versa

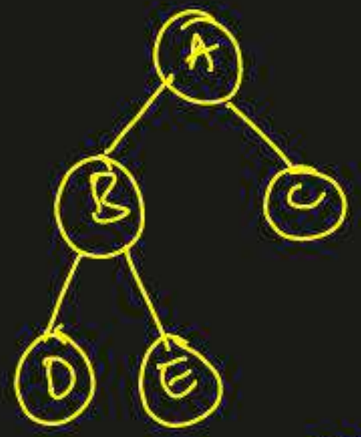
Perfect Binary Tree

FBT, with all leaf nodes at same level.

A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.

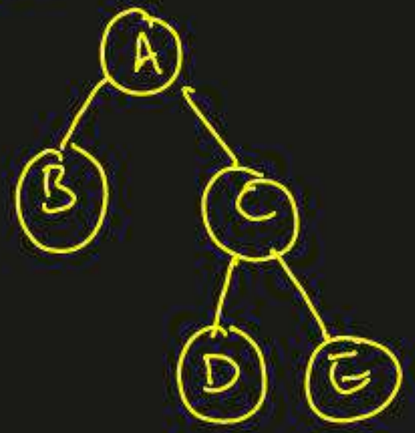


FBT, CBT, PBT

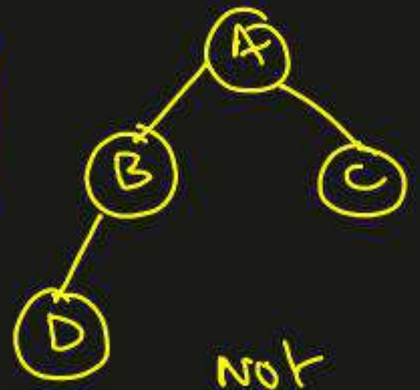


FBT, CBT,

Not PBT
C at level 1
D, E at level 2



FBT but
Not CBT
Not PBT



CBT, Not FBT, Not PBT

Skewed / Degenerate Binary Tree

All internal nodes, either only left child or only right child.

The degenerate binary tree is a tree in which all the internal nodes have only one children.

2 Forms : Left skewed and Right skewed.

Left skewed

Right Skewed.

Skewed / Degenerate Binary Tree

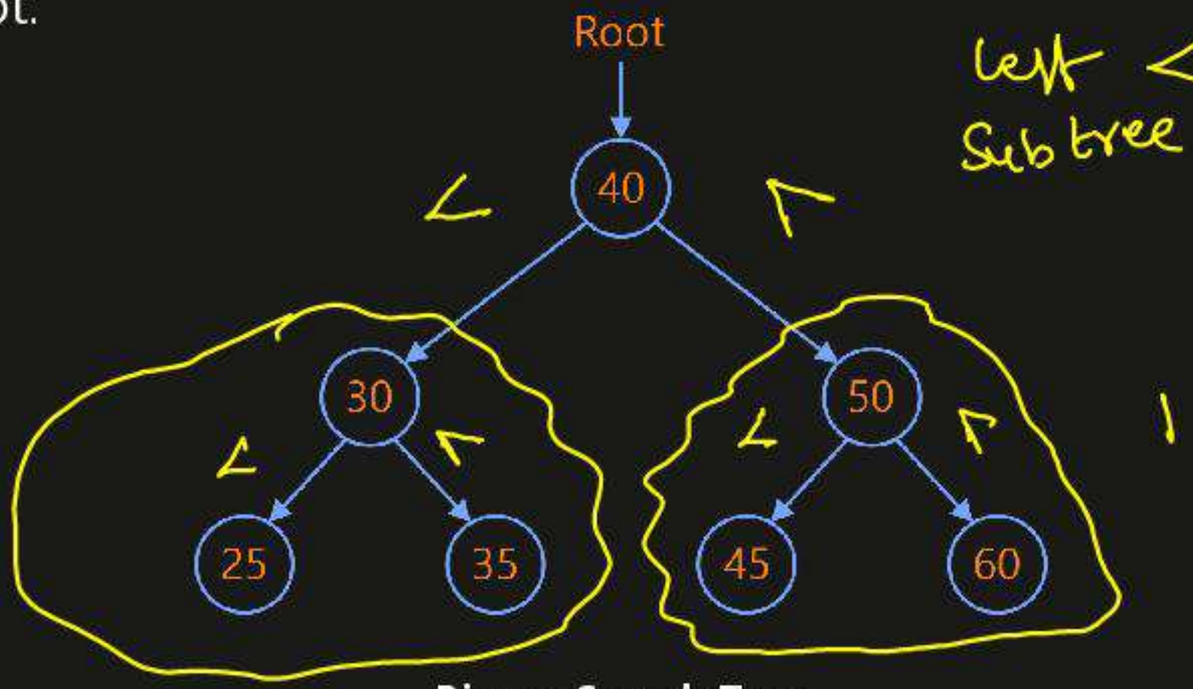
The degenerate binary tree is a tree in which all the internal nodes have only one children.

BYJU'S EXAM PREP

Binary Search Tree

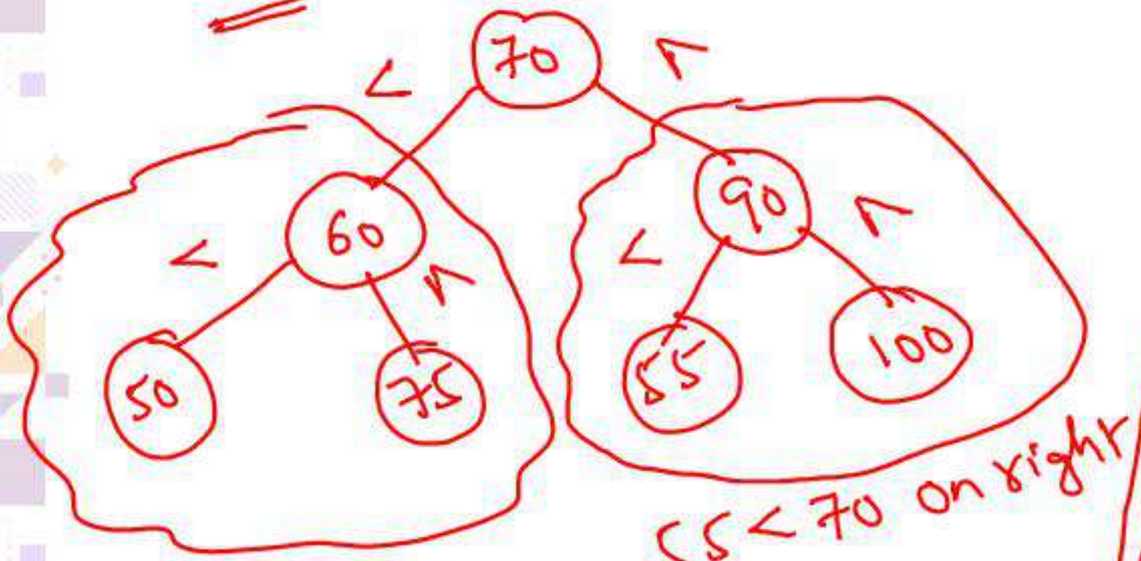
In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

Left Sub tree < Parent < Right Sub tree



Binary Search Tree

Ex:1

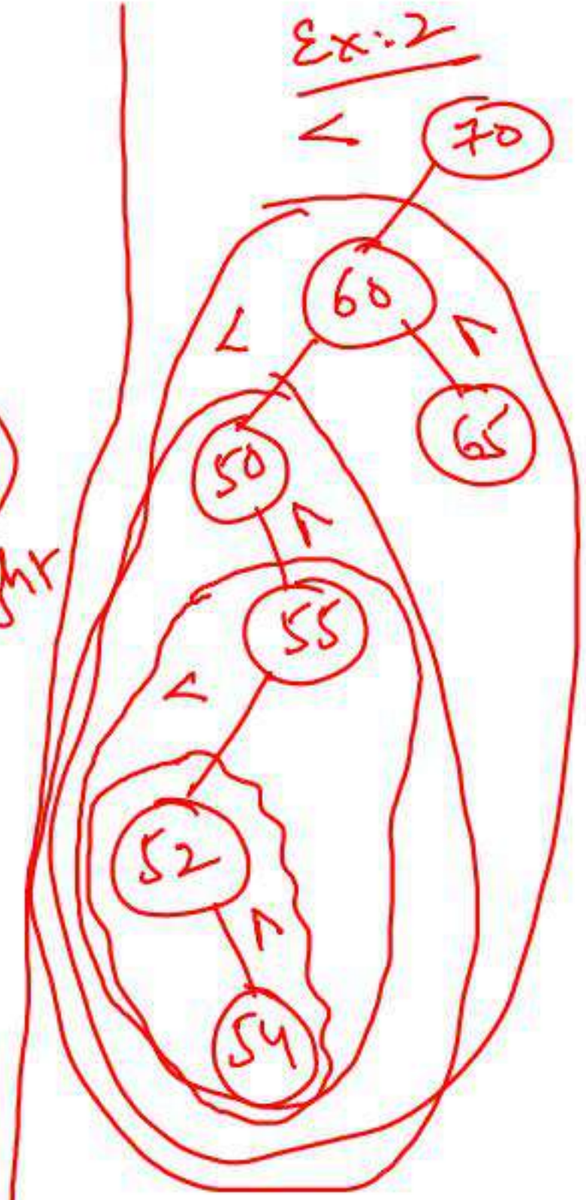


55 < 70 on right

75 > 70 on left

Not a BST.

Ex:2



Binary search Tree.

BYJU'S EXAM PREP

BST - Operations

The Operations that can be performed on BST include :

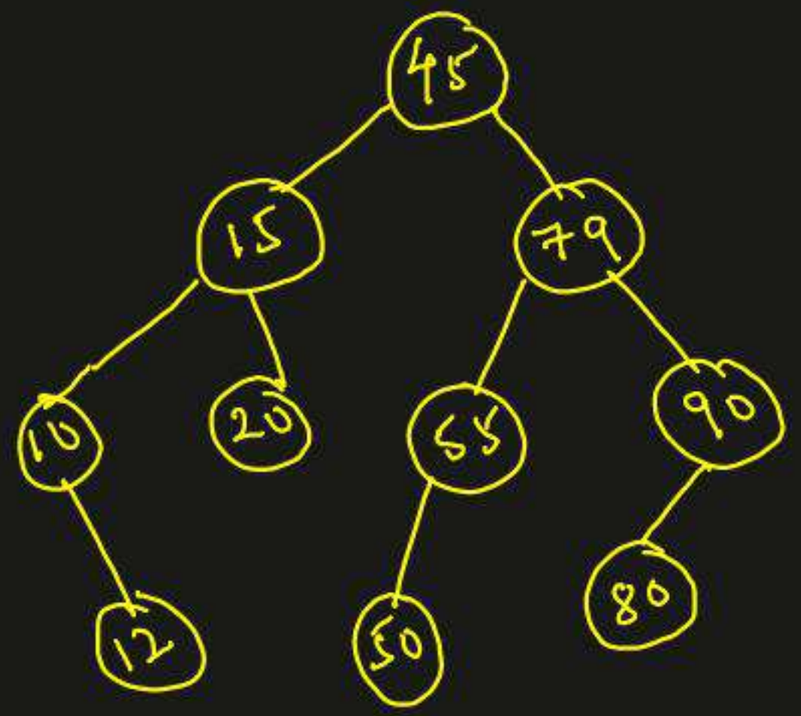
- Construction
- Insertion
- Deletion
- Search

BYJU'S EXAM PREP

BST – Construction / Insertion

Root node.

Construct a BST with elements 45, 15, 79, 90, 10, 55, 12, 20, 50 inserted in the same sequence.



Insert 80

80 > 45, 80 > 79, 80 < 90

15 < 45

79 > 45

90 > 45, 90 > 79

10 < 45, 10 < 15

55 > 45, 55 < 79

12 < 45, 12 < 15, 12 > 10

20 < 45, 20 > 15

50 > 45, 50 < 79, 50 < 55

BYJU'S EXAM PREP

Deletion from BST

In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations that may occur :

- The node to be deleted is the leaf node, or, (Zero children) : simply delete
- The node to be deleted has only one child, and,
- The node to be deleted has two children

BYJU'S EXAM PREP

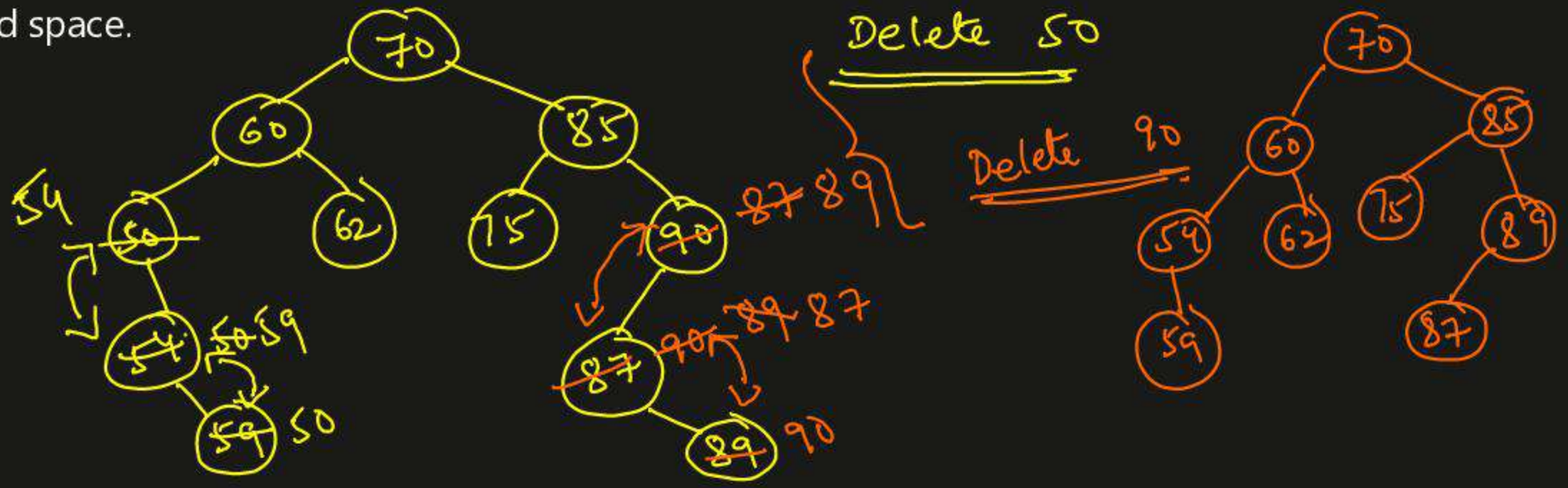
Deletion from BST – Leaf Node

It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.

BYJU'S EXAM PREP

Deletion from BST – Node With 1 Child

In this case, we have to replace the target node with its child, and then delete the child node. It means that after replacing the target node with its child node, the child node will now contain the value to be deleted. So, we simply have to replace the child node with NULL and free up the allocated space.



BYJU'S EXAM PREP

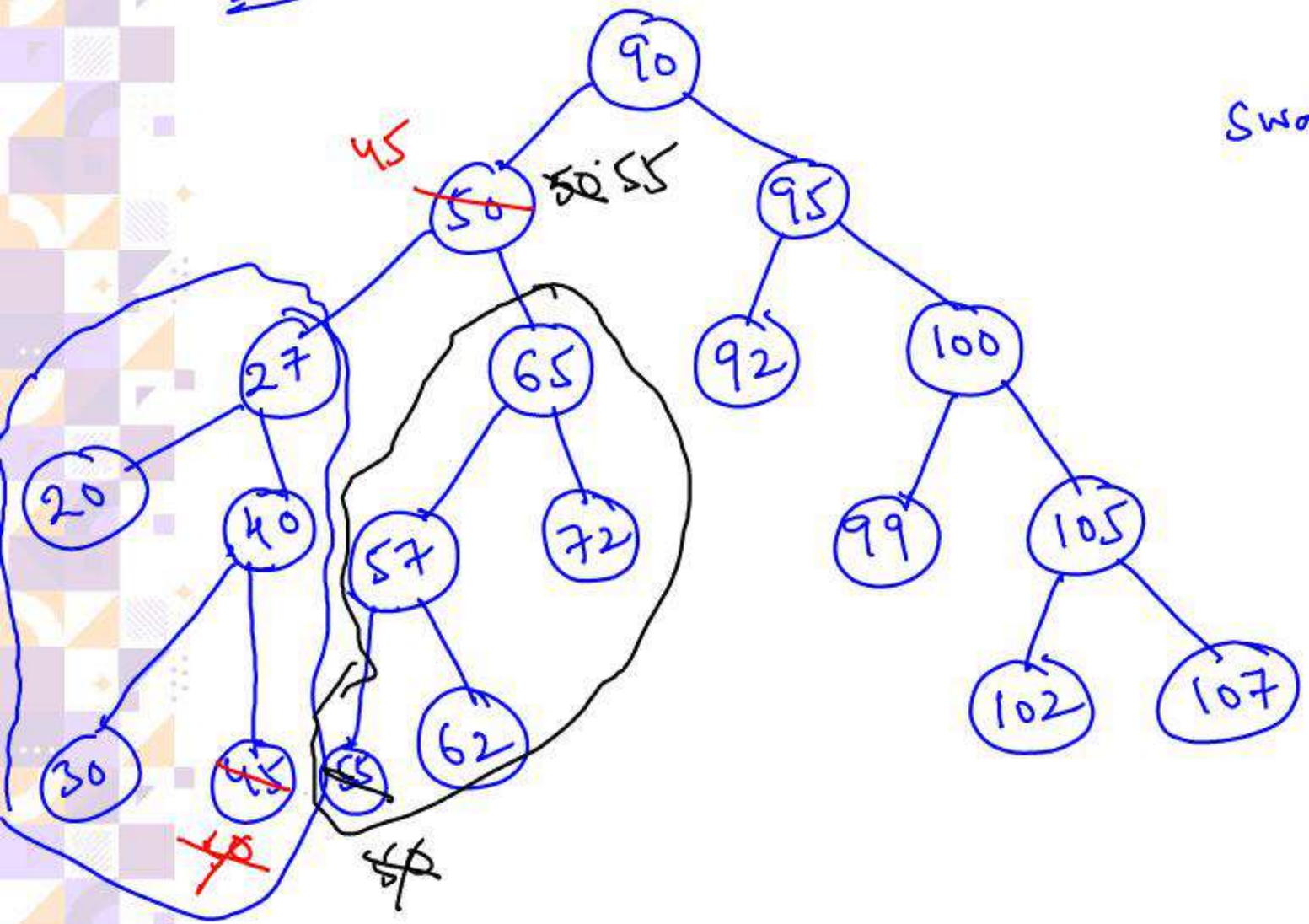
Deletion from BST – Node With 2 Children

Swap with Biggest of left subtree
(OR)
Swap with smallest of right sub tree.

This case of deleting a node in BST is a bit complex among other two cases. In such a case, the steps to be followed are listed as follows -

- First, find the inorder successor of the node to be deleted.
- After that, replace that node with the inorder successor until the target node is placed at the leaf of tree.
- And at last, replace the node with NULL and free up the allocated space.

Ex:



Delete 50

Swap with Biggest of left subtree

50 & 45 are swapped,

Delete 50 as it is leaf.

(OR)

Swap with smallest of right subtree.

50 & 55 are swapped.

Then Delete 50 as it is leaf node.

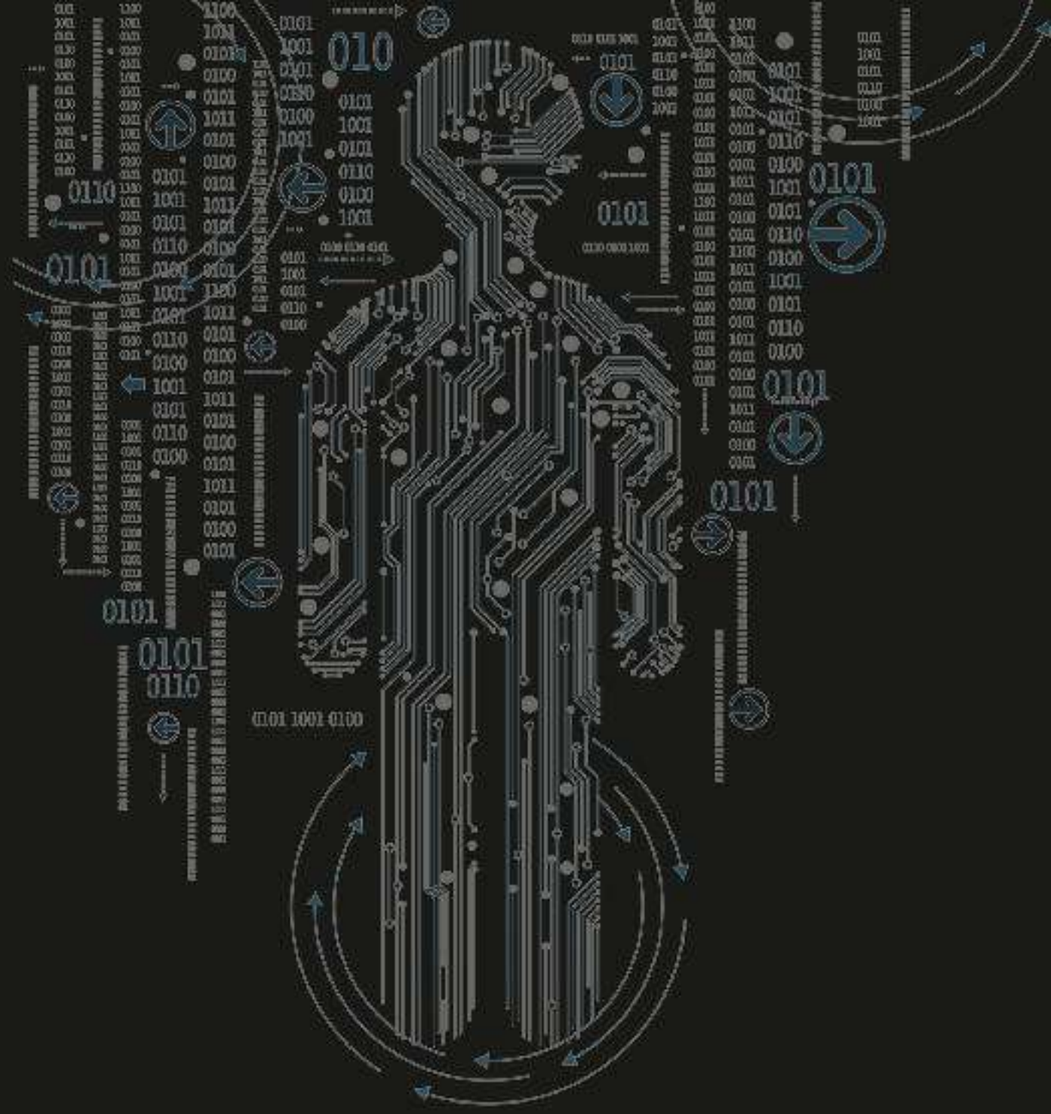
Tomorrow @ 10 AM.

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Tree Traversals



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Trees : Fundamentals of Binary Trees — Types of Trees

– Binary Search Tree : Left Subtree < Parent < Right Subtree

– Construction, Insertion, Deletion, Search

- Leaf Node
- Node with 1 child
- Node with 2 children

Queues : Dequeue
- i/p restricted
- o/p restricted

Priority Queue : Introduced

BYJU'S EXAM PREP

f(Q) i=1 → f(Q) i=2 → f(Q) i=3 → f(Q) i=4, f=r=-1

f=1, f=2, f=3, r=f=4

insert(2) insert(3) insert(4)

r=0 4, r=1 3, r=2 2, r=3 1

Q4. Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are: [GATE 2007]

i. isEmpty (Q) — returns true if the queue is empty, false otherwise.

ii. delete (Q) — deletes the element at the front of the queue and returns its value.

iii. insert (Q, i) — inserts the integer i at the rear of the queue.

What operation is performed by the above function f ?

Consider the following function:

```
void f (queue Q) {
  int i ;
  if (!isEmpty(Q)) {
     i = delete(Q);
     f(Q);
     insert(Q, i);
  }
}
```

A. Leaves the queue Q unchanged

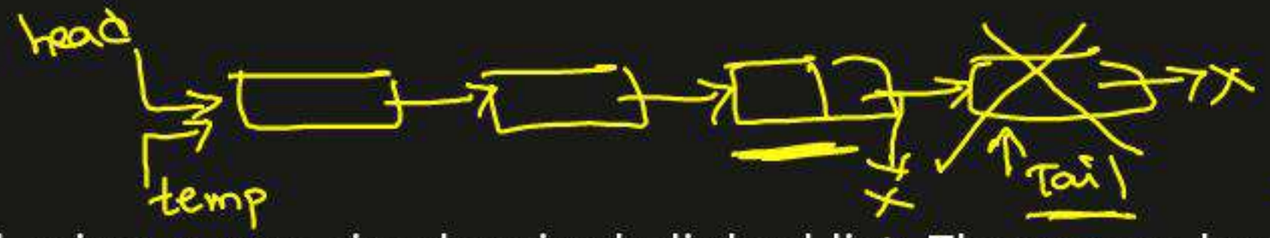
B. Reverses the order of the elements in the queue Q

C. Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

D. Empties the queue Q

BYJU'S EXAM PREP

New = head
head = New } O(1)



free(Tail)

Q5. A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.

Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure ?

[insert at beginning] [delete last node] in SLL.

[GATE 2018]

A. Θ(1), Θ(1)

B. Θ(1), Θ(n)

C. Θ(n), Θ(1)

D. Θ(n), Θ(n)

Delete last node:
(n-1) nodes traversed.
O(n-1)
= O(n) - O(1) = O(n)

Tree Traversals

Traversal

- The order of visiting/Accessing nodes in a Binary tree is called as Traversal.
- 2 types of Traversals:

  (a) Breadth First Traversal (Level Order Traversal)

  (b) Depth First Traversal

  (i) In order

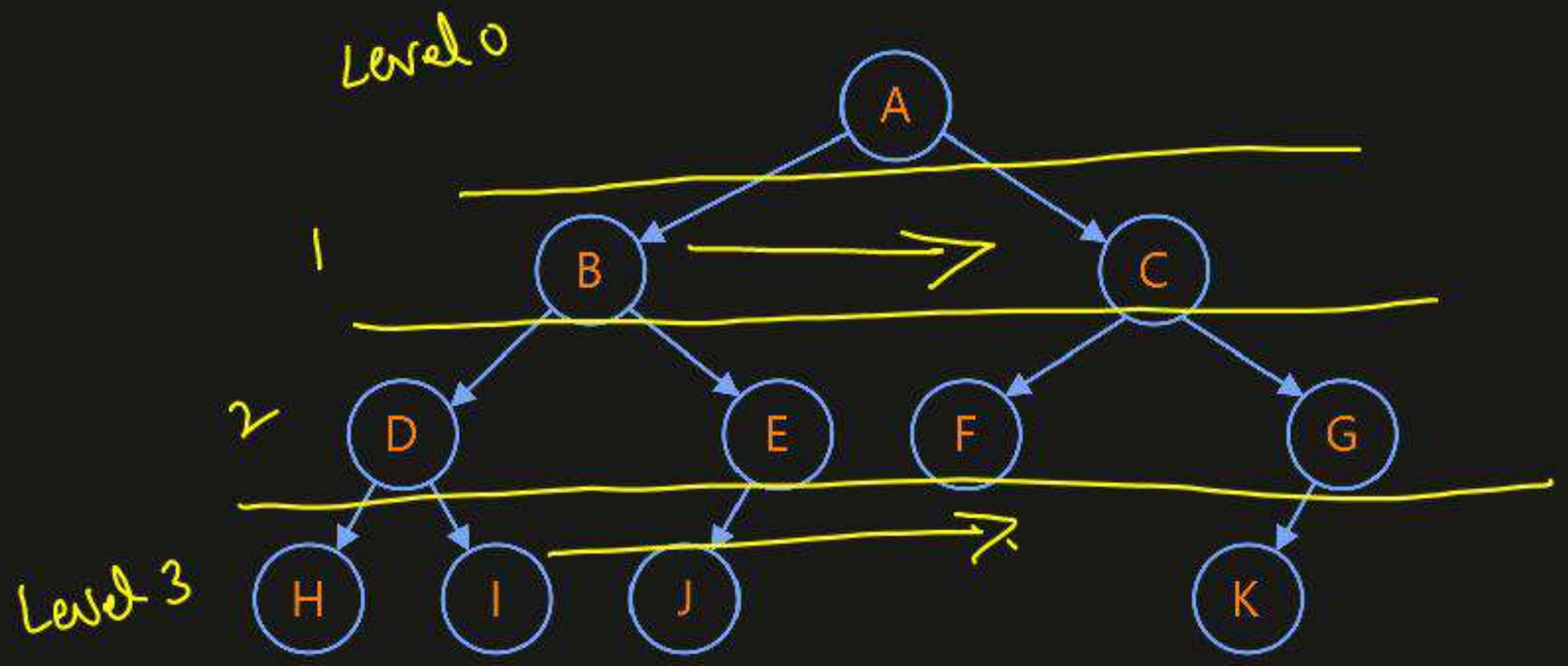
  (ii) Pre order

  (iii) Post order

BYJU'S EXAM PREP

Breadth – First Traversal

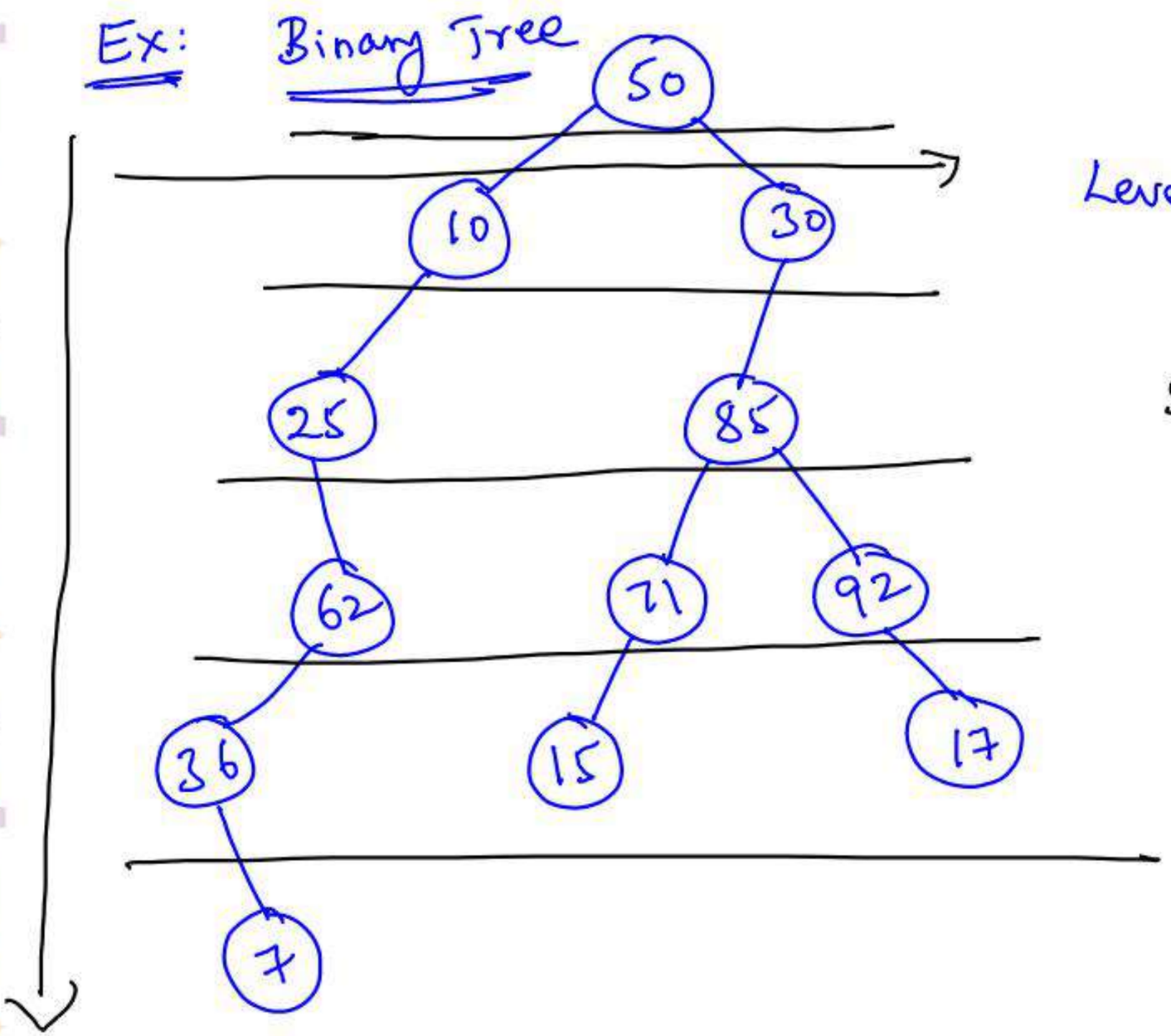
Top level to bottom level, left to Right.



Breadth – First Traversal

A, B, C, D, E, F, G, H, I, J, K

Ex: Binary Tree



Level order Traversal sequence:

50, 10, 30, 25, 85, 62,

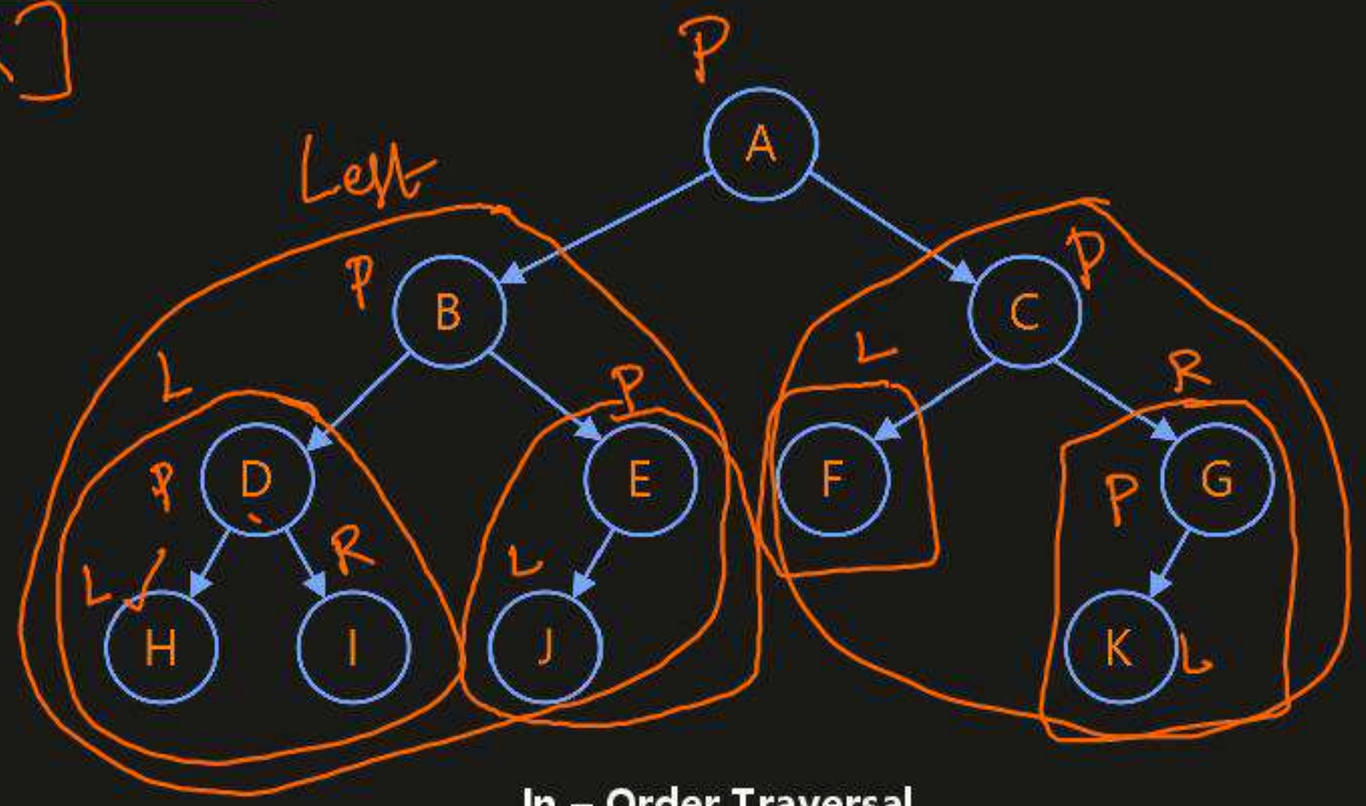
71, 92, 36, 15, 17, 7

BYJU'S EXAM PREP

In – Order Traversal

Left Parent Right

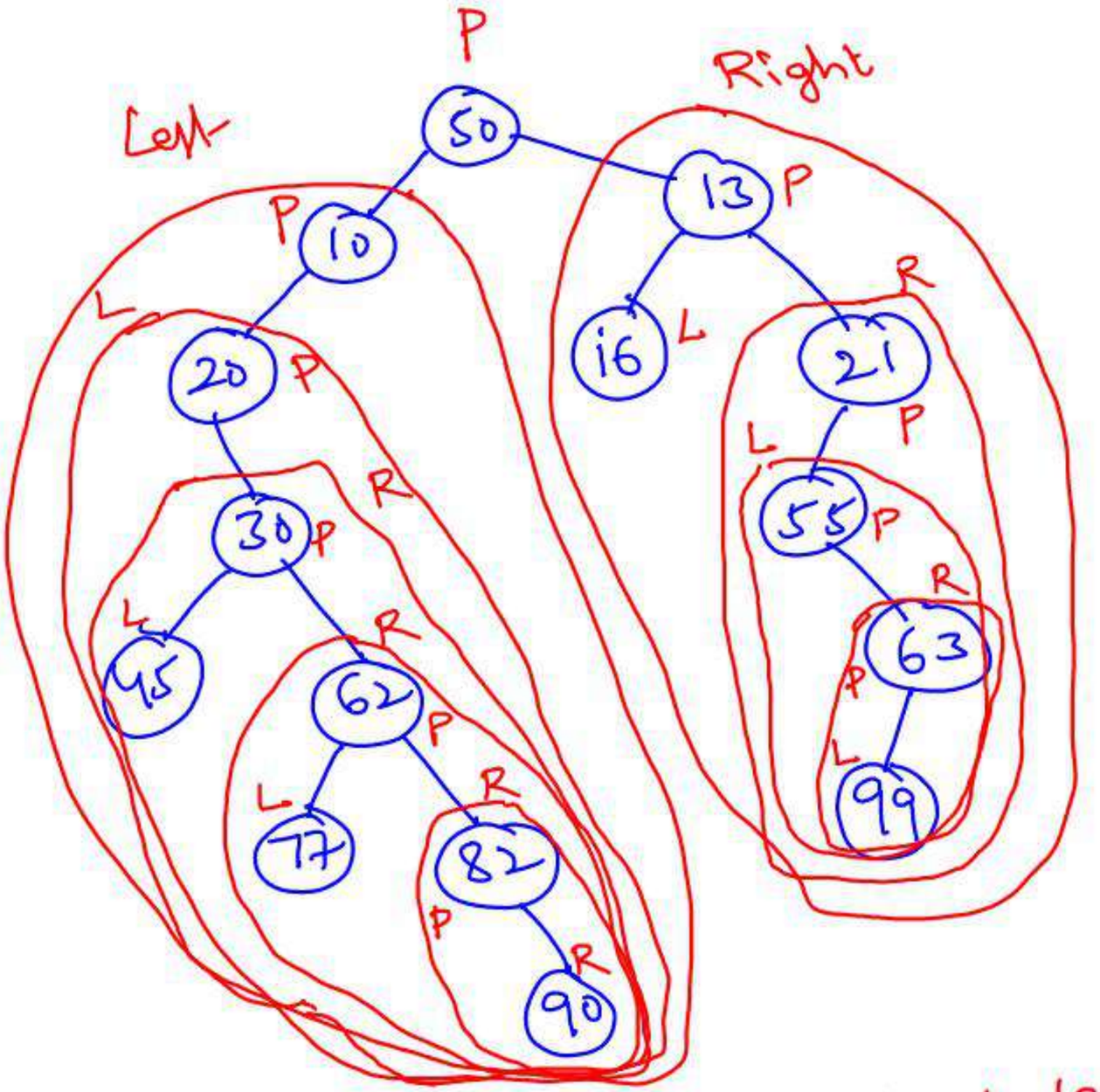
[L P R]

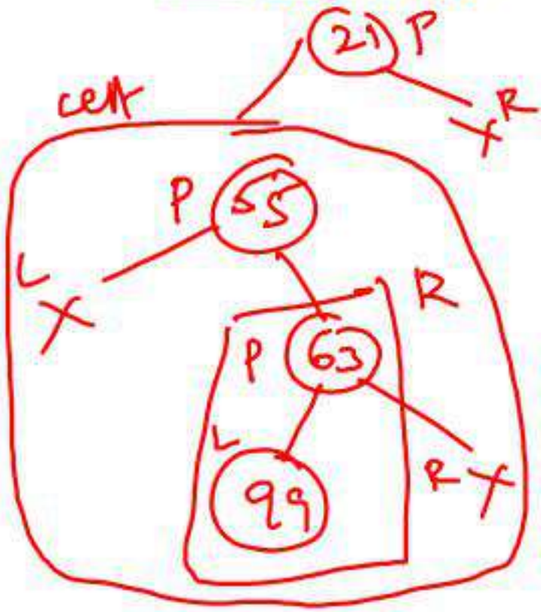


In – Order Traversal

In order : H D I B J E A F C K G

Ex: 2





55, 99, 63, 21

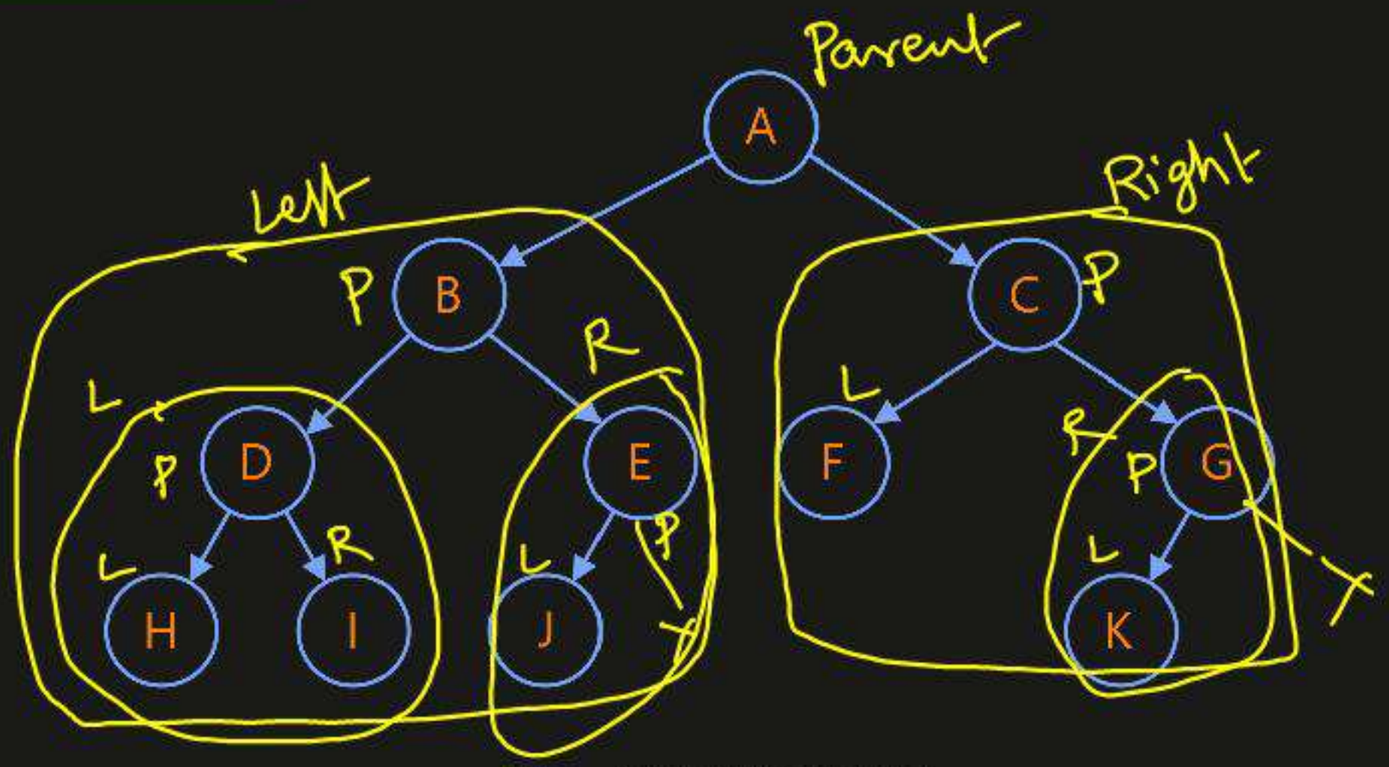
In order : 20, 45, 30, 77, 62, 82, 90, 10, 50, 16, 13, 55, 99, 63, 21

Tree Traversals

Pre – Order Traversal

Parent, left, Right

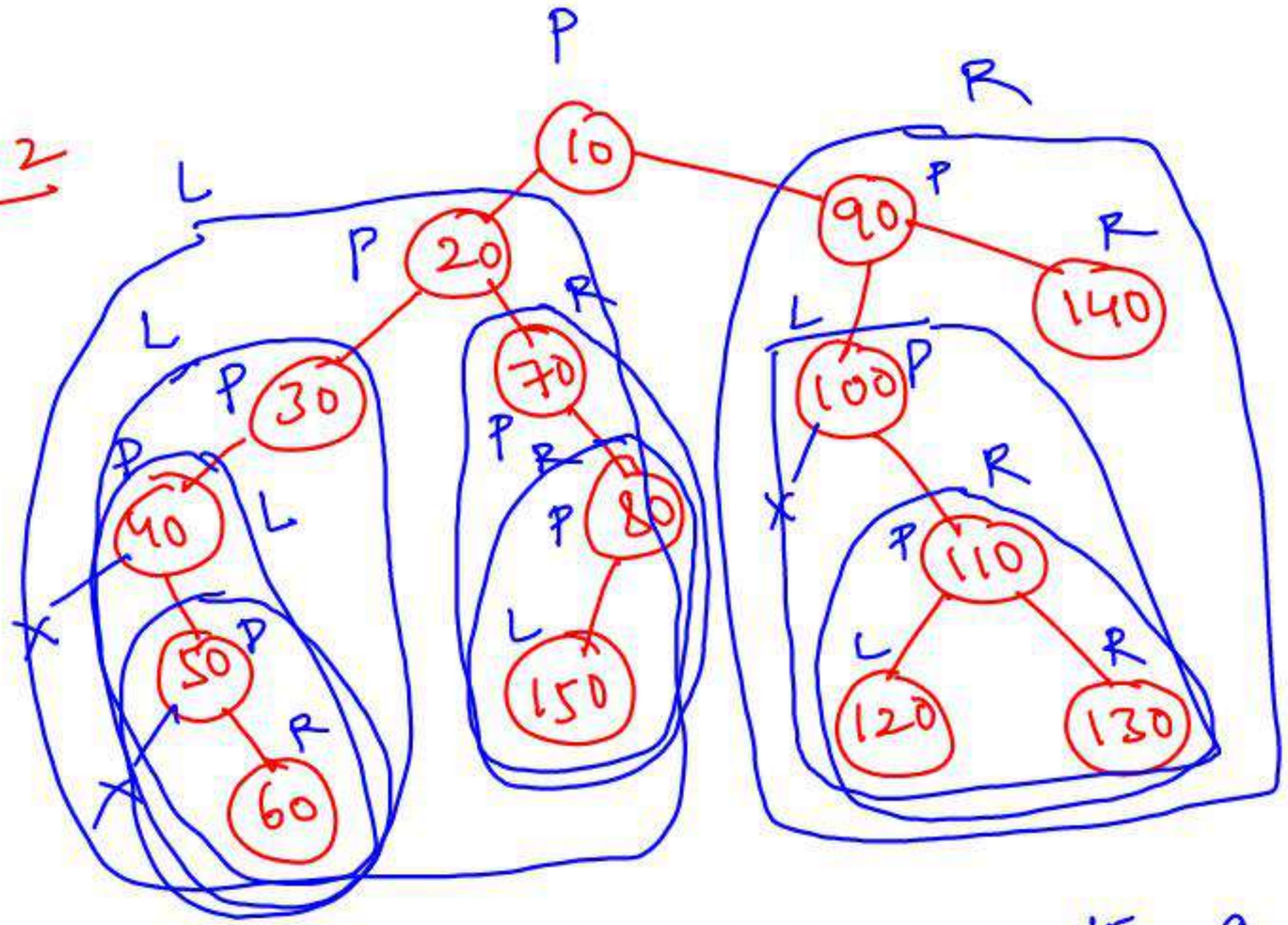
P L R



Pre – Order Traversal

Pre order : A, B, D, H, I, E, J, C, F, G, K

Ex: 2



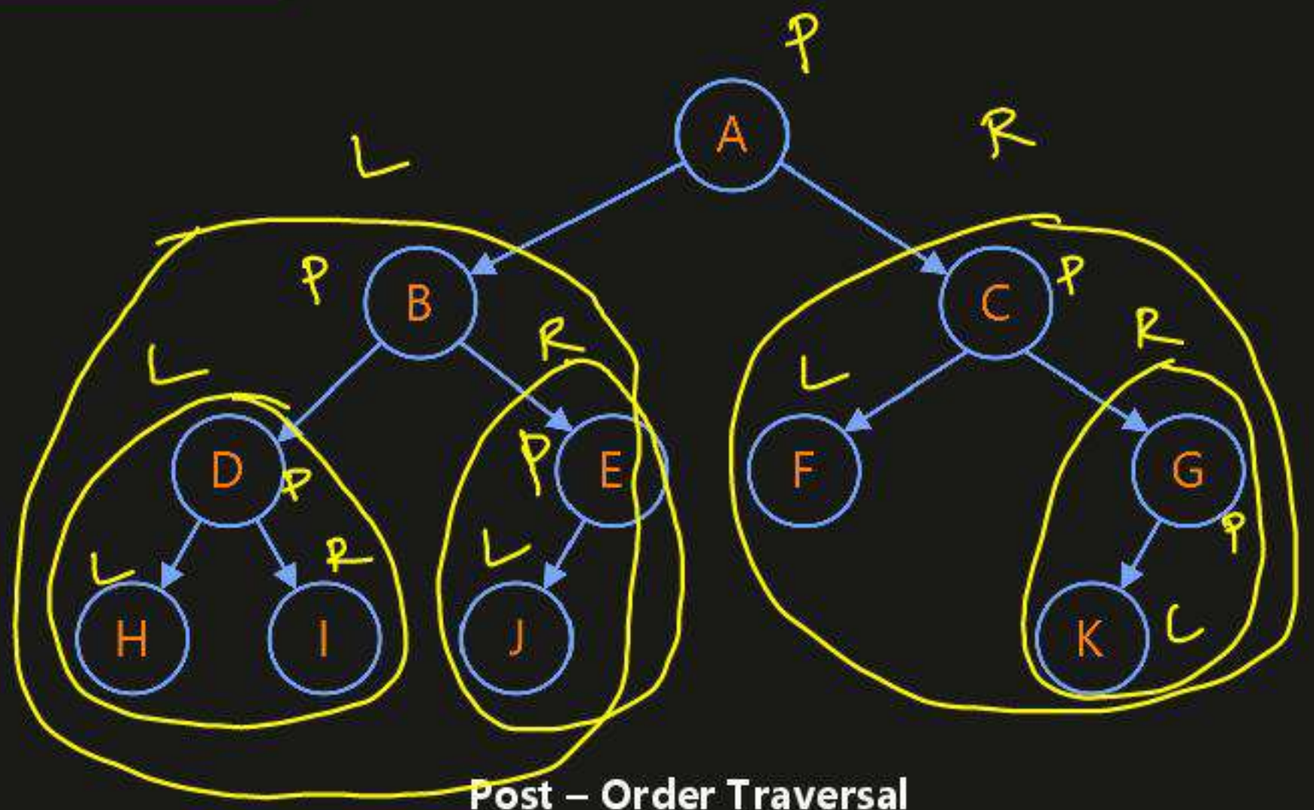
Pre order : 10, 20, 30, 40, 50, 60, 70, 80, 150, 90, 100, 110, 120, 130, 140

BYJU'S EXAM PREP

Post – Order Traversal

Left Right Parent

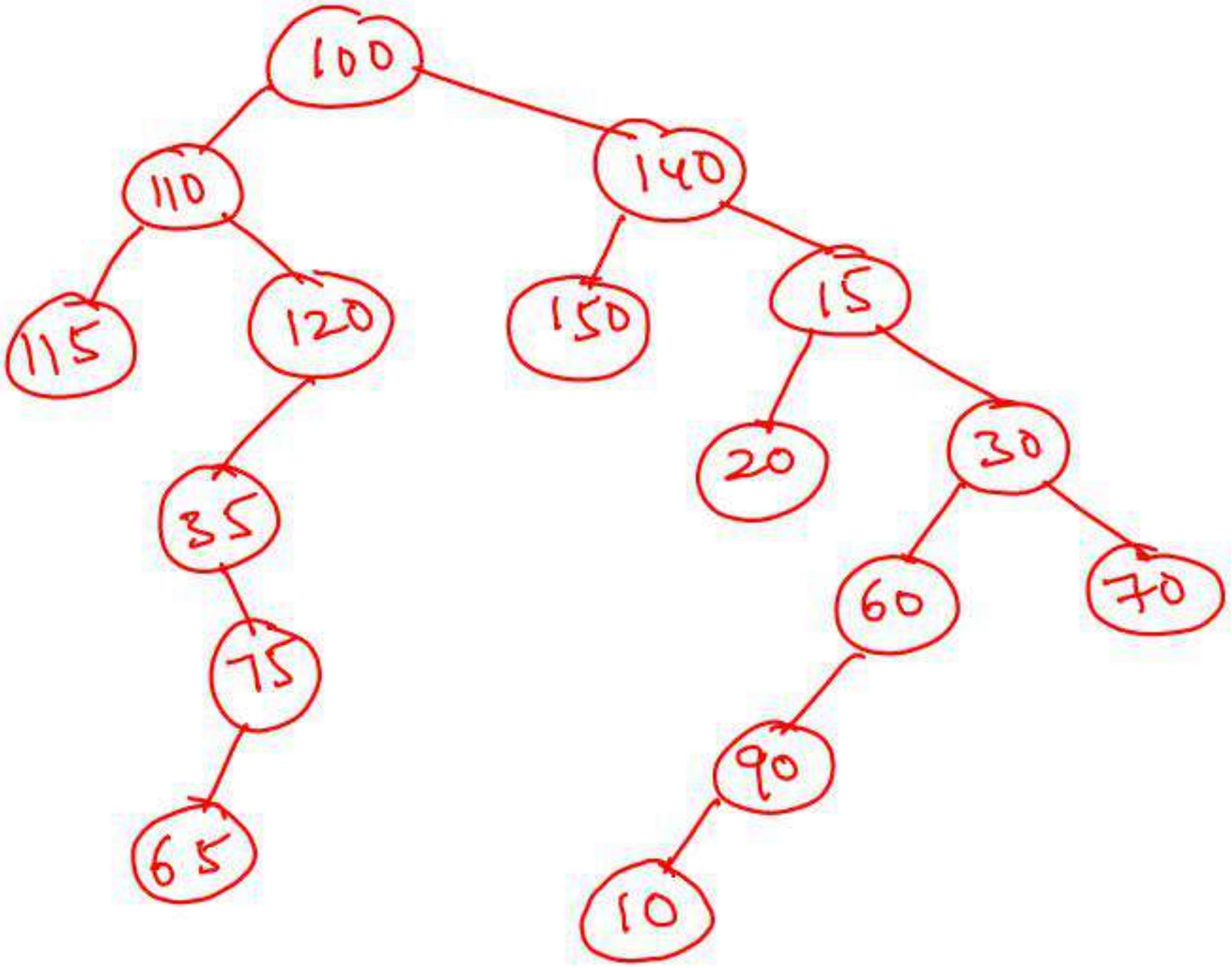
L R P



Post – Order Traversal

H, I, D, J, E, B, F, K, G, C, A

Ex: 2

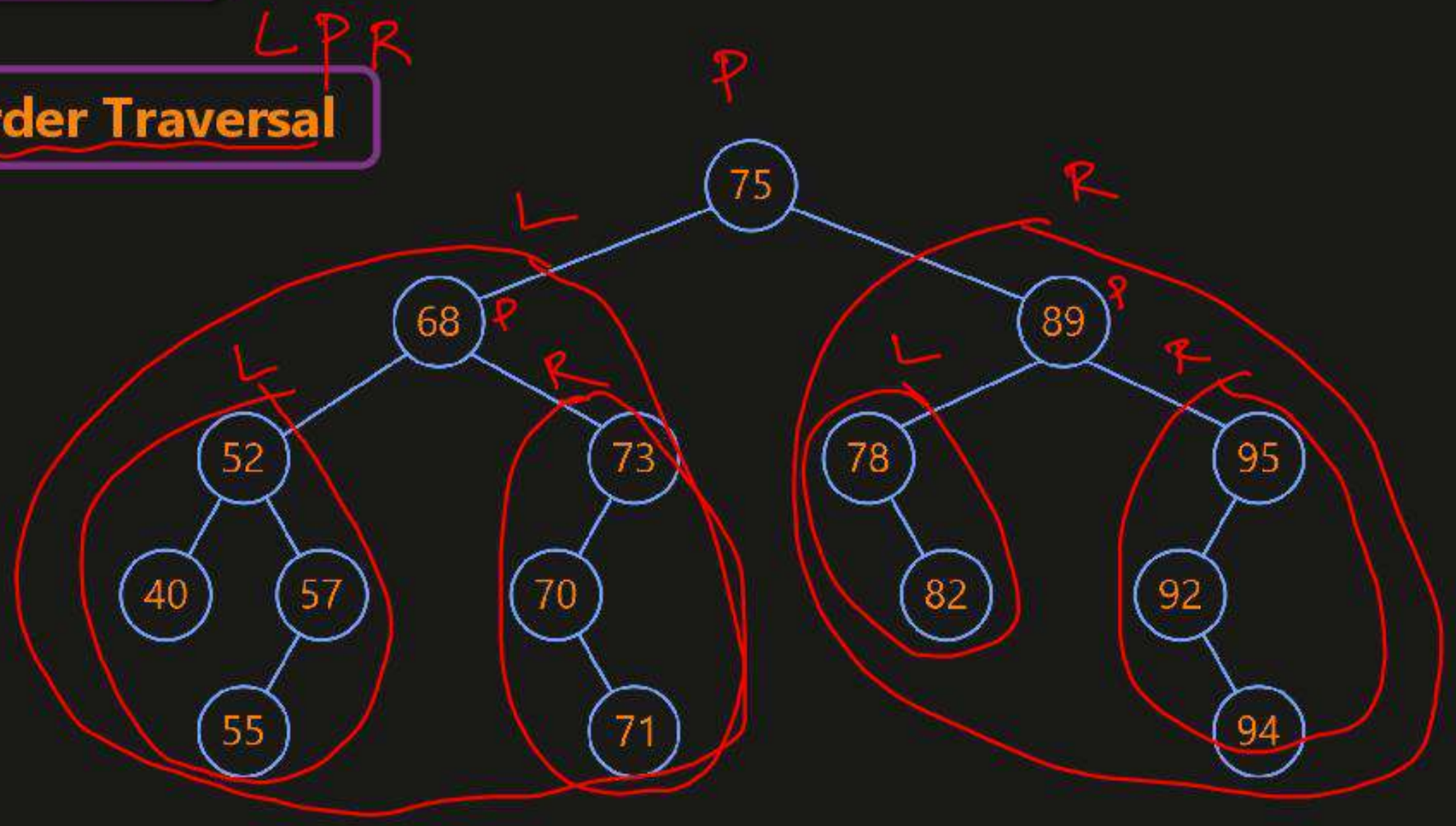


Post order : 115, 65, 75, 35, 120, 110, 150, 20, 10, 90, 60, 70, 30, 15, 140, 100

BYJU'S EXAM PREP

BST – In Order Traversal

Example: BST



BST – In Order Traversal

In order Traversal of BST is always in the ascending order of values.

40, 52, 55, 57, 68, 70, 71, 73, 75, 78, 82, 89, 78, 92, 94, 95.

Tree Traversals

Practice Problem - 1

Level order:

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

In order: (LPR)

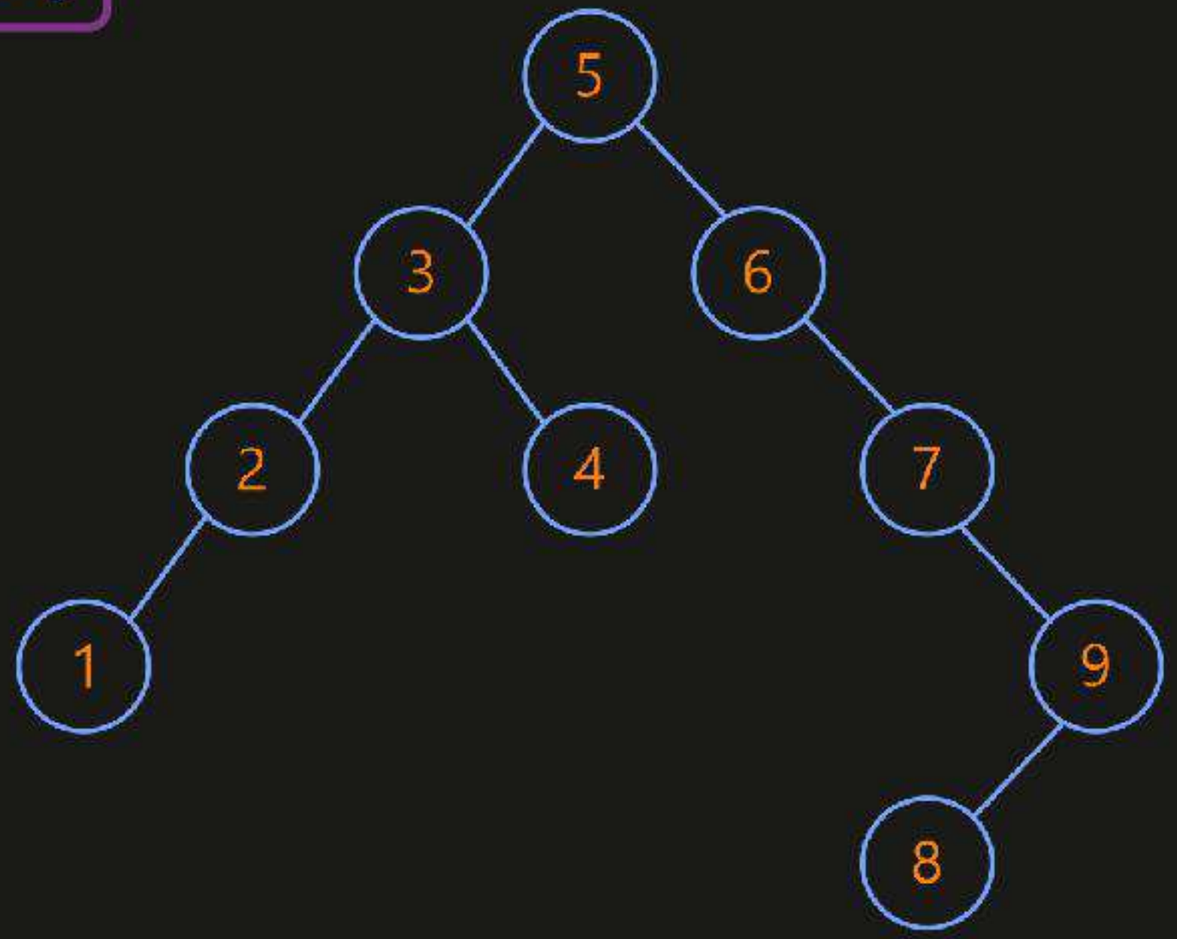
1, 2, 3, 4, 5, 6, 7, 8, 9

Pre order: (PLR)

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

Post order: (LRP)

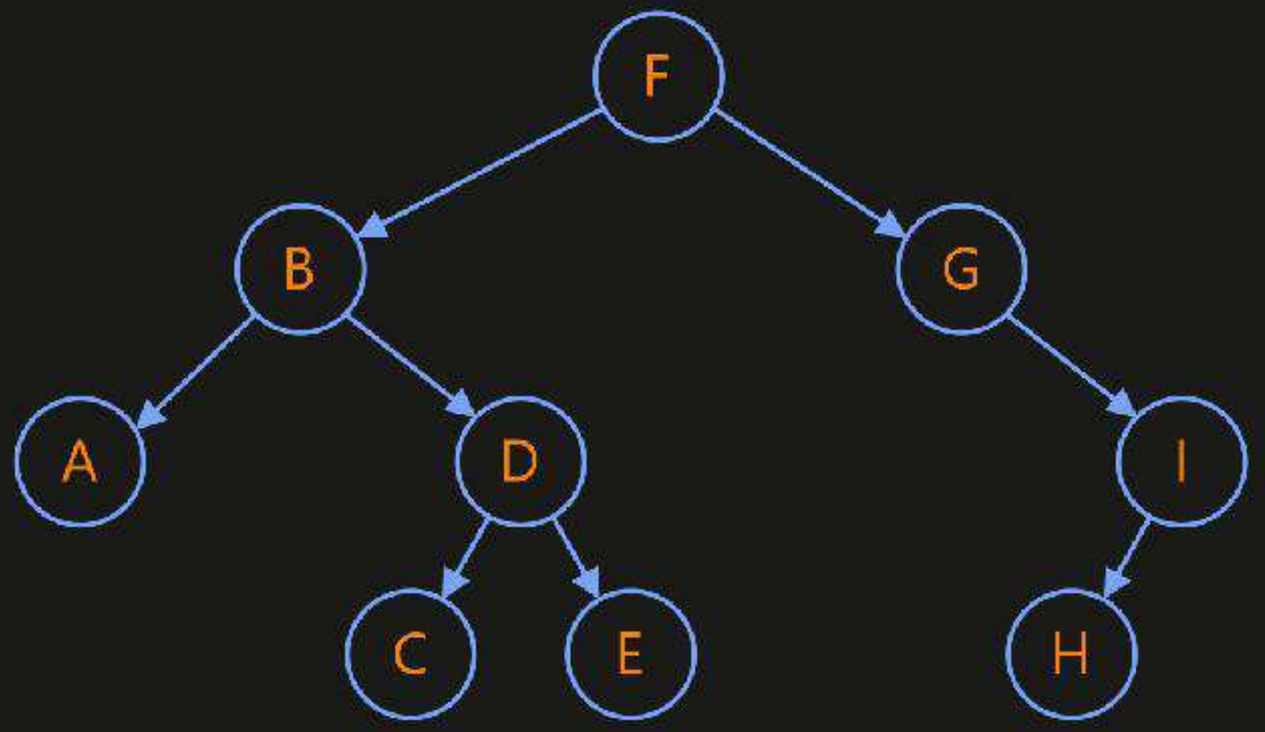
1, 2, 4, 3, 8, 9, 7, 6, 5



BYJU'S EXAM PREP

Practice Problem - 2

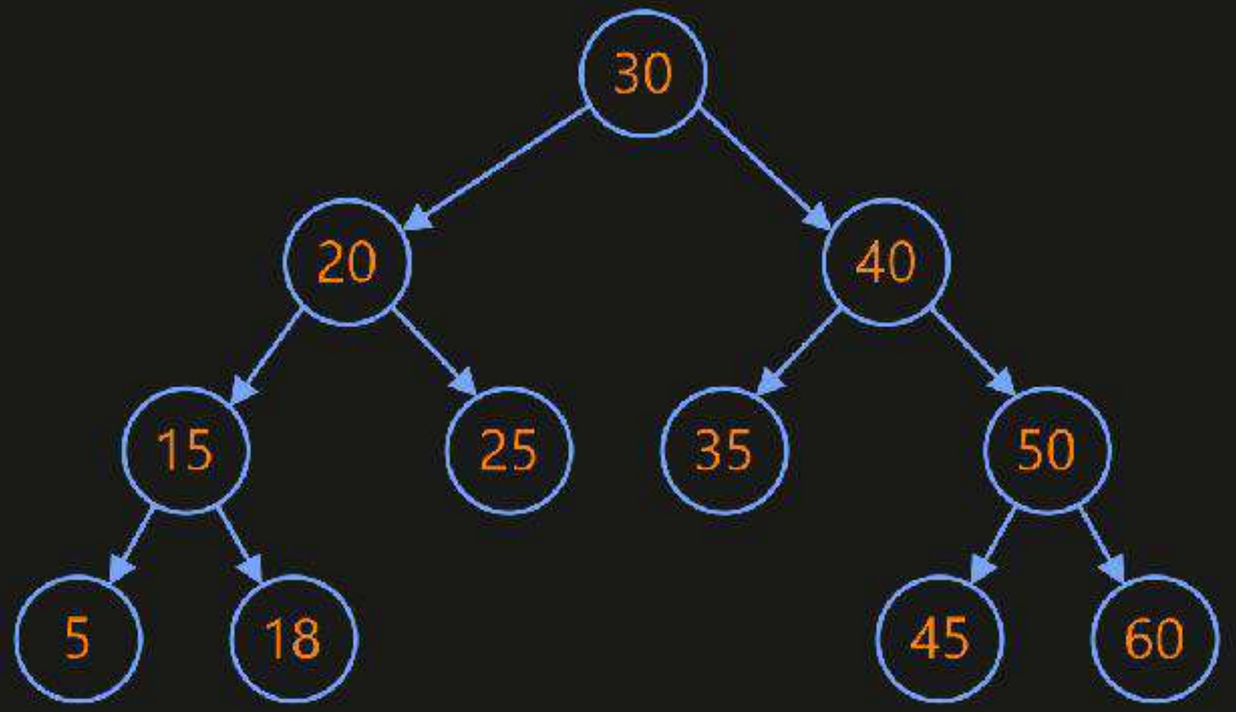
Home Work



BYJU'S EXAM PREP

Practice Problem - 3

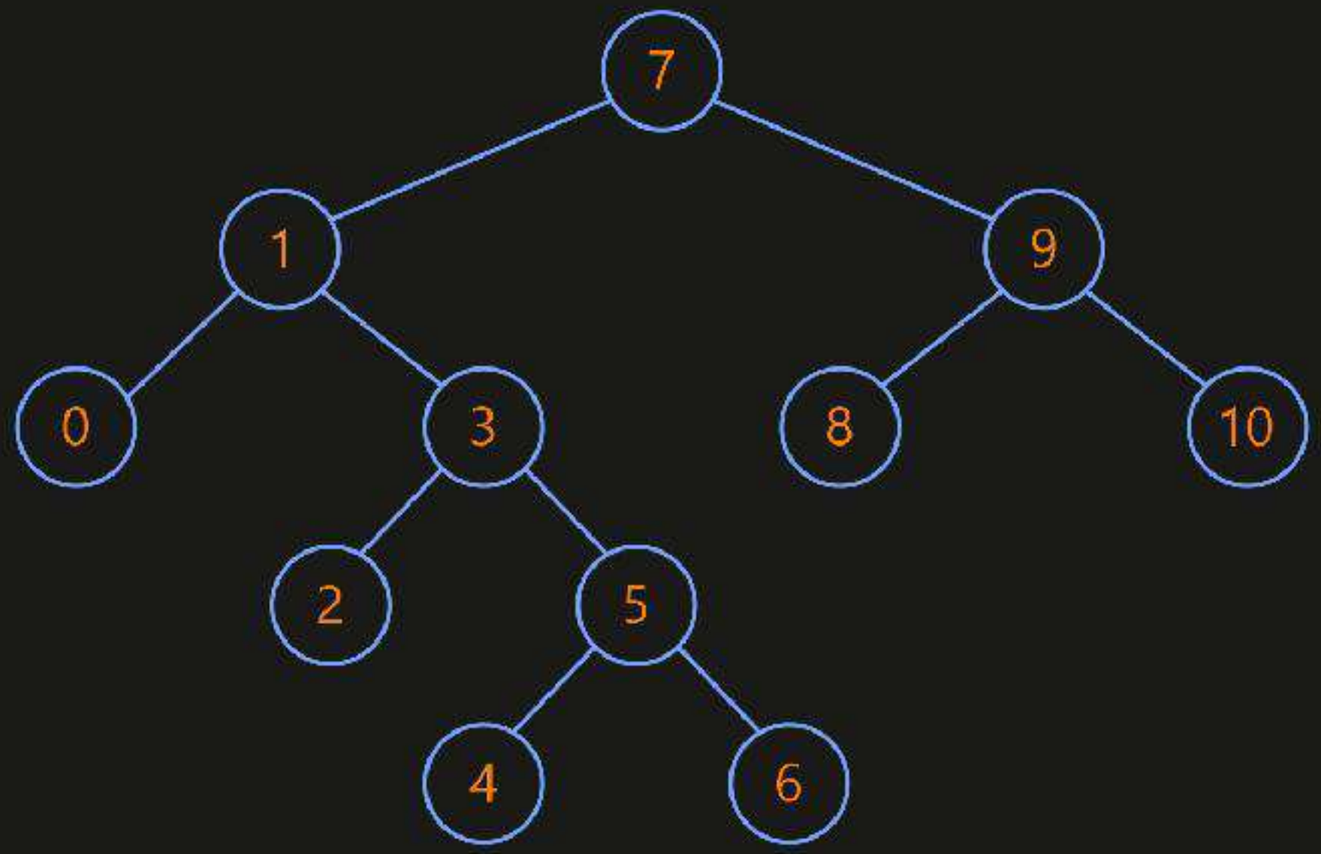
Home work



BYJU'S EXAM PREP

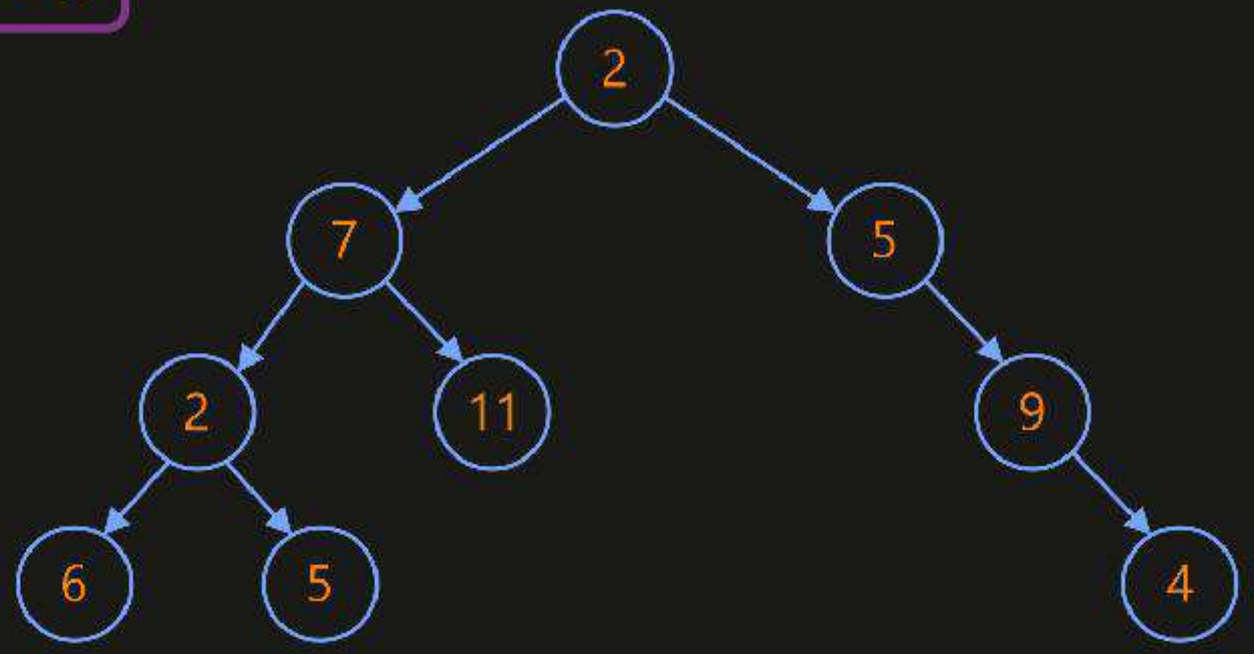
Practice Problem - 4

Home work



Practice Problem - 5

Home work



Tree Traversals

Program in 'C'

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
  int item;
  struct node* left;
  struct node* right;
};

// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}

// preorderTraversal traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
```

Tree Traversals

Program in 'C'

```c
// postorderTraversal traversal
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}

int main() {
  struct node* root = createNode(1);
  insertLeft(root, 12);
  insertRight(root, 9);

  insertLeft(root->left, 5);
  insertRight(root->left, 6);

  printf("Inorder traversal \n");
  inorderTraversal(root);

  printf("\nPreorder traversal \n");
  preorderTraversal(root);

  printf("\nPostorder traversal \n");
  postorderTraversal(root);
}
```

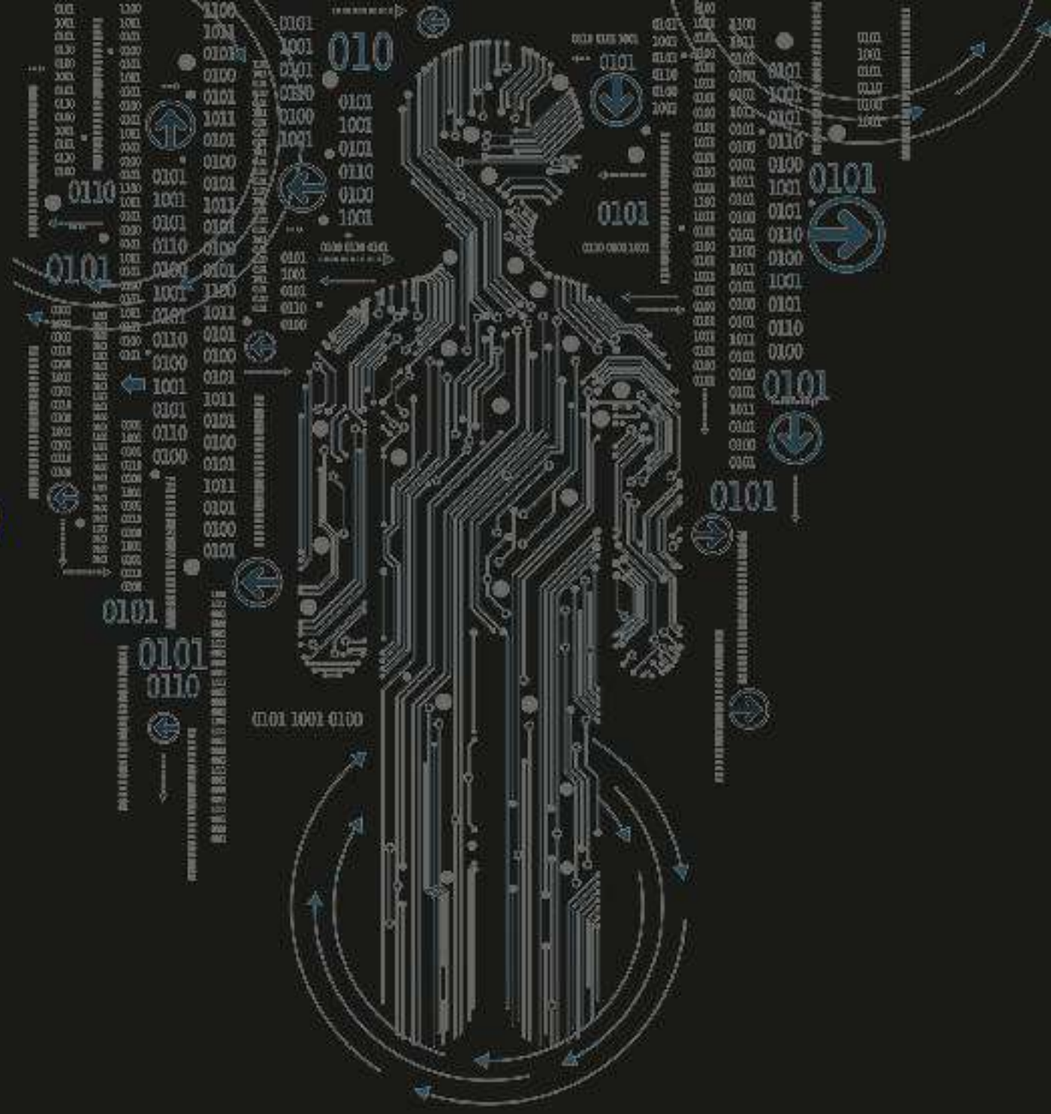
BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

NOTE: Tomorrow 25/4/22 – No classes
(will be rescheduled)

PROGRAMMING & DATA STRUCTURES

Binary Heap



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

Tree Traversals :- Breadth-First Traversal (Level-order)

- Depth-First Traversal
  - In-order (L P R)
  - Pre-order (P L R)
  - Post-order (L R P)

L = left P = Parent R = Right

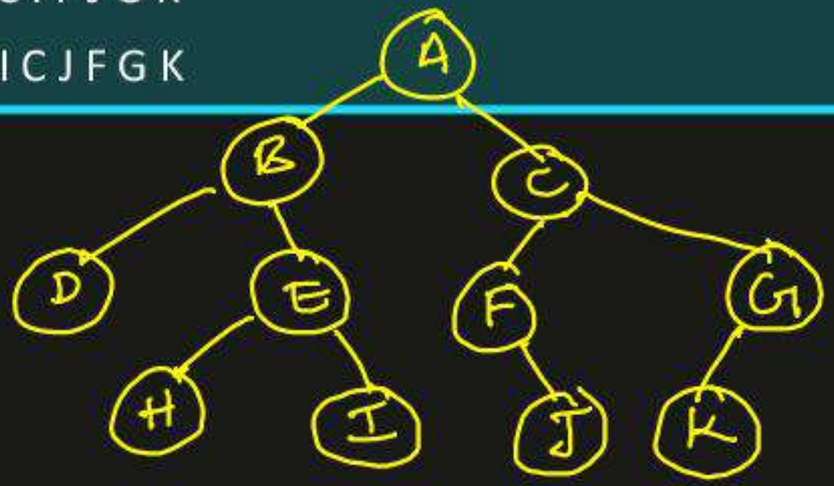
Binary Heap

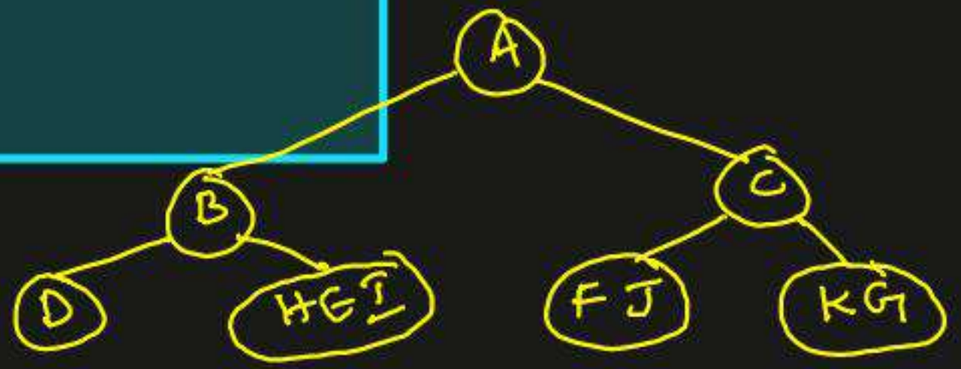
In order : L P R
Pre order : P L R

Post order : L R P

For a Binary Tree, The In order Traversal Sequence is D B H E I A F J C K G and Post Order Sequence is D H I E B J F K G C A. The Pre Order Traversal Sequence is _____

A. A B D H E I C F J G K
B. A B D E H I C F J G K
C. A B D E H C I F J G K
D. A B D E H I C J F G K





Pre order: A B D E H I C F J G K

Binary Heap

Pre order : P L R

For a Binary Tree, The In order Traversal Sequence is C D B A F H G I E J and Pre Order Sequence is A B C D E F G H I J. The Post Order Traversal Sequence is _____

A. D C B H G I F J E A

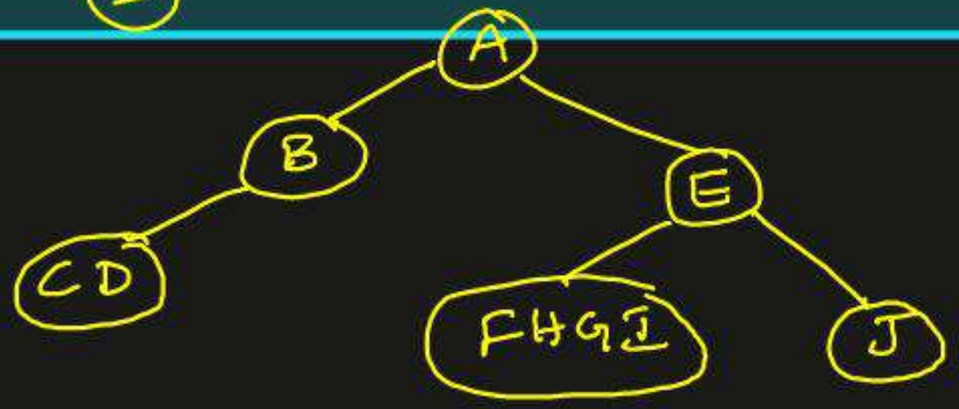
B. D C B H I G J F E A

C. D C B H I G F J E A

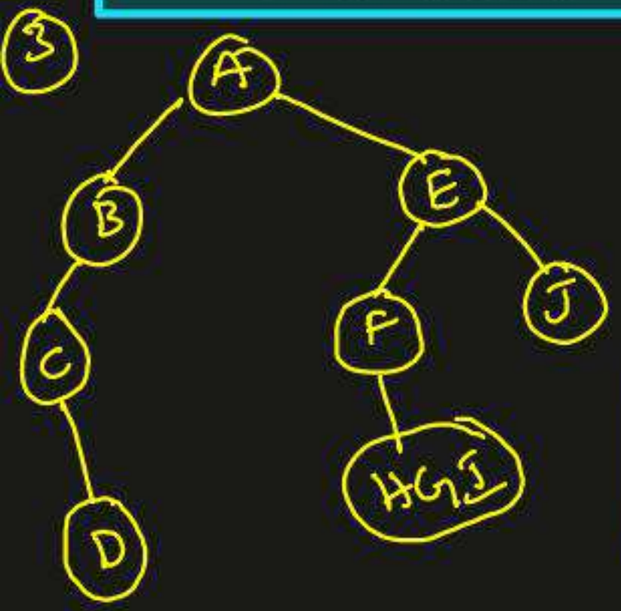
D. D C B H G F I J E A

1

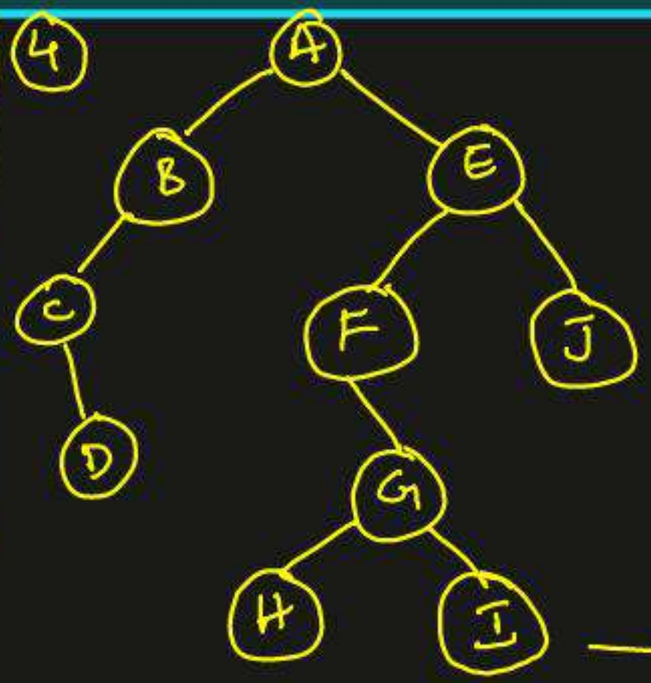
2



3



4



→ Post order: D C B H I G F J E A

Binary Heap

The Pre Order traversal sequence for the BST with Post order sequence as 42,66,69,68,65,76,90,82,73 is _____

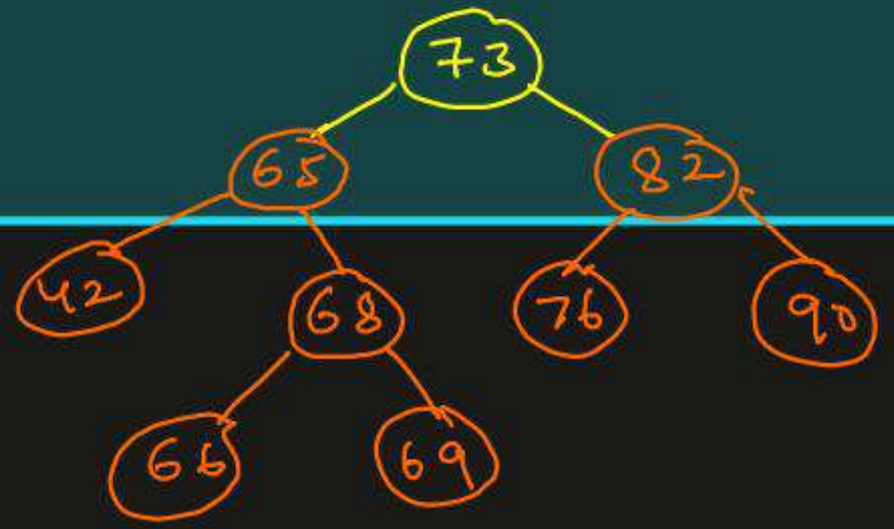
A. 73,65,42,68,66,69,76,82,90

B. 73,65,42,68,69,66,76,82,90

C. 73,65,42,68,66,69,82,76,90

D. 73,65,42,66,68,69,76,82,90

Inorder : 42, 65, 66, 68, 69, 73, 76, 82, 90



Pre order : 73, 65, 42, 68, 66, 69, 82, 76, 90
(PLR)

BYJU'S EXAM PREP

Binary Heap

- A binary heap is a heap data structure that takes the form of a binary tree.
- Binary heaps are a common way of implementing priority queues.
- The binary heap was introduced by J. W. J. Williams in 1964.

BYJU'S EXAM PREP

Binary Heap

A binary heap is defined as a binary tree with two additional constraints :

Structuring Property.

- Shape property: a binary heap is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.

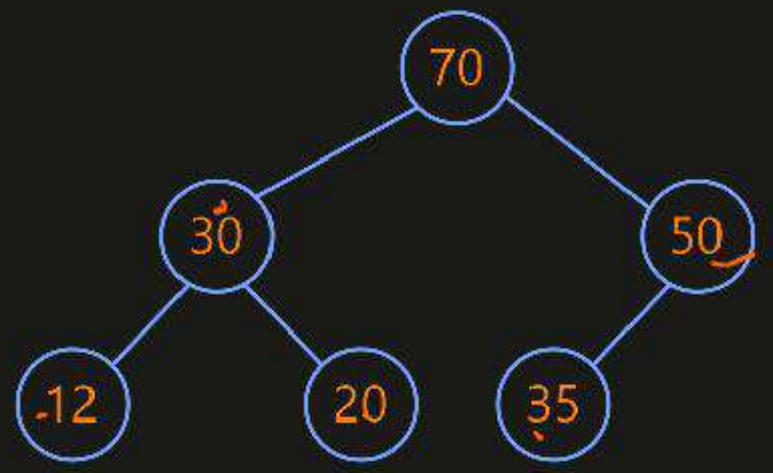
Ordering Property

- Heap property: the key stored in each node is either greater than or equal to (≥) or less than or equal to (≤) the keys in the node's children.
- Heaps where the parent key is greater than or equal to (≥) the child keys are called max-heaps; those where it is less than or equal to (≤) are called min-heaps.

Max - Heap

In this heap, the key value of a node is greater than or equal to the key value of the highest child.

Hence, H[Parent(i)] ≥ H[i]

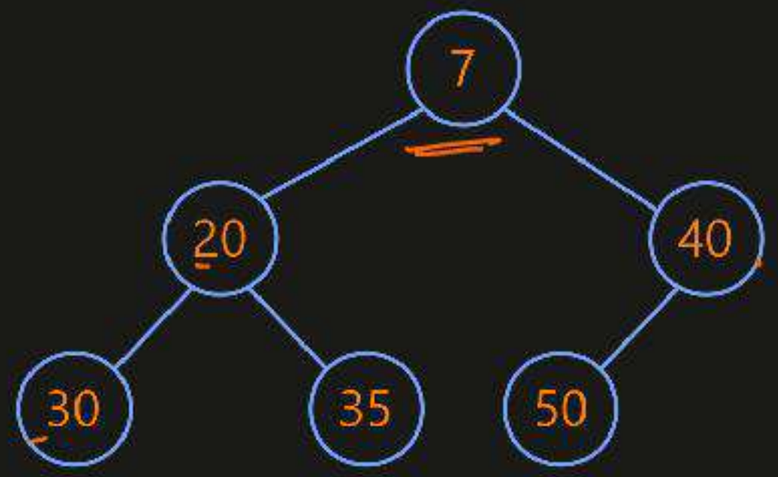


Max - Heap

Min - Heap

In mean-heap, the key value of a node is lesser than or equal to the key value of the lowest child.

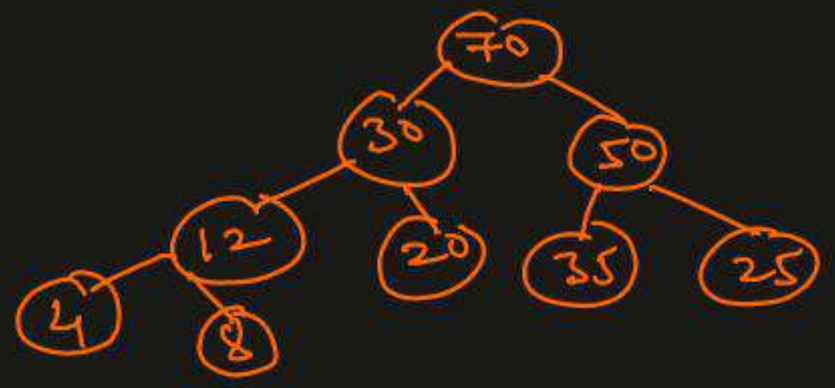
Hence, H[Parent(i)] ≤ H[i]



Min - Heap

BYJU'S EXAM PREP

Array Representation



- A complete binary tree can be represented by an array, storing its elements using level order traversal.
- Let us consider a heap (as shown below) which will be represented by an array H.

Index	0	1	2	3	4	5	6	7	8	...
Elements	70	30	50	12	20	35	25	4	8	...

Binary Heap

Array Representation

- If a parent at index 'i', then it's left child at index 2* i and right child at index (2 * i +1).
- Indexing from 1 would be more understandable.

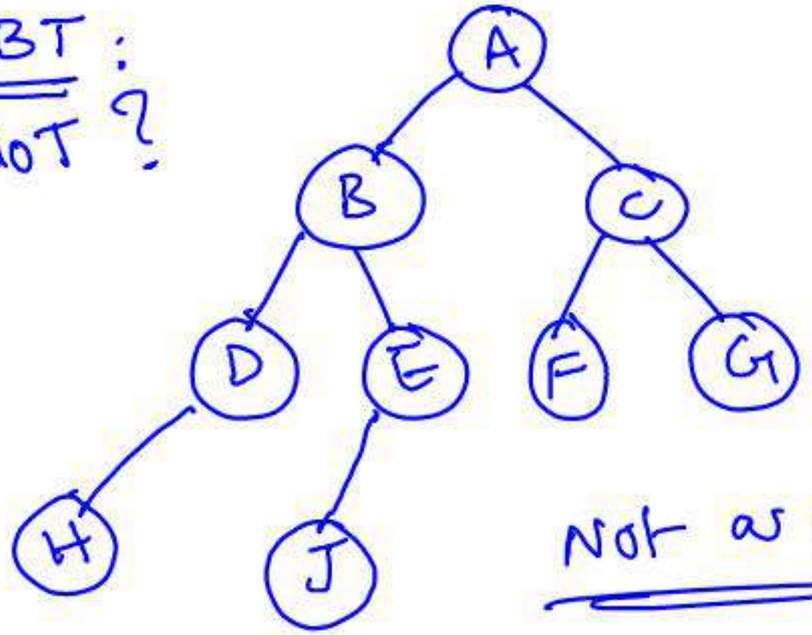
2(1) 2(1)+1

Index	1	2	3	4	5	6	7	8	9	...
Elements	70	30	50	12	20	35	25	4	8	...

i=2 2(2) 2(2)+1

i=3

CBT : OR NOT ?



Parent : 'i' index

left child : 2*i

Right child : (2*i) +1

Not a CBT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
A	B	C	D	E	F	G	H	—	J	—	—	—	—	—	—

In between, if gap, Not CBT.

gap at End is ok

BYJU'S EXAM PREP

Building Max Heap

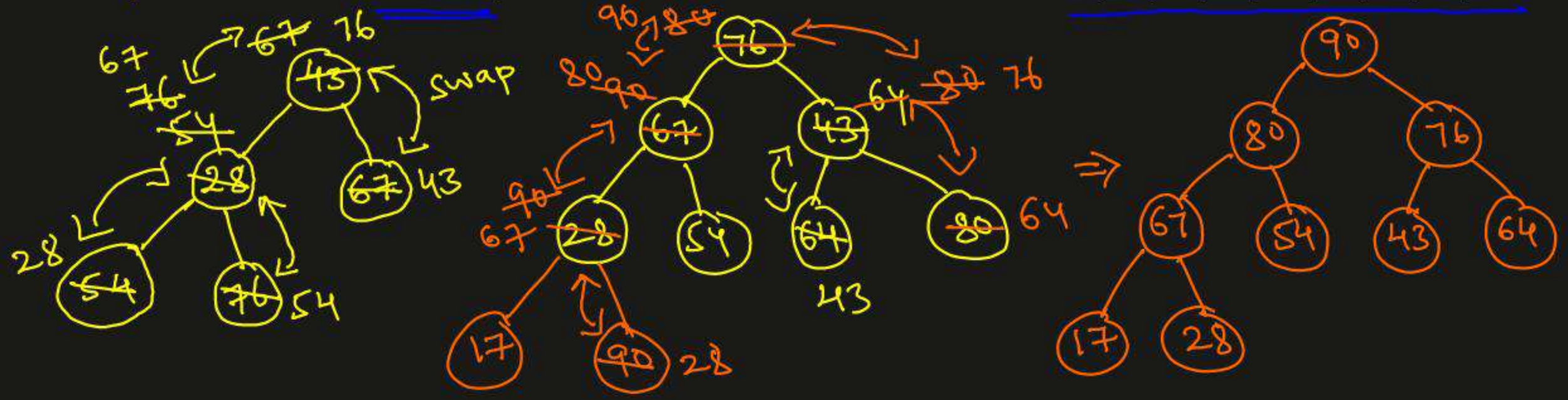
- First, we have to insert the element in such a way that the property of the complete binary tree must be maintained.
- Secondly, the value of the parent node should be greater than the either of its child.

BYJU'S EXAM PREP

Building Max Heap

Compare
1) Parent < child
2) if not swap
} Heapification.

Example: Construct a Max heap with elements inserted in the order: 43, 28, 67, 54, 76, 64, 80, 17, 90.



Level order of Max-Heap: 90, 80, 76, 67, 54, 43, 64, 17, 28

BYJU'S EXAM PREP

Insertion into Max-Heap

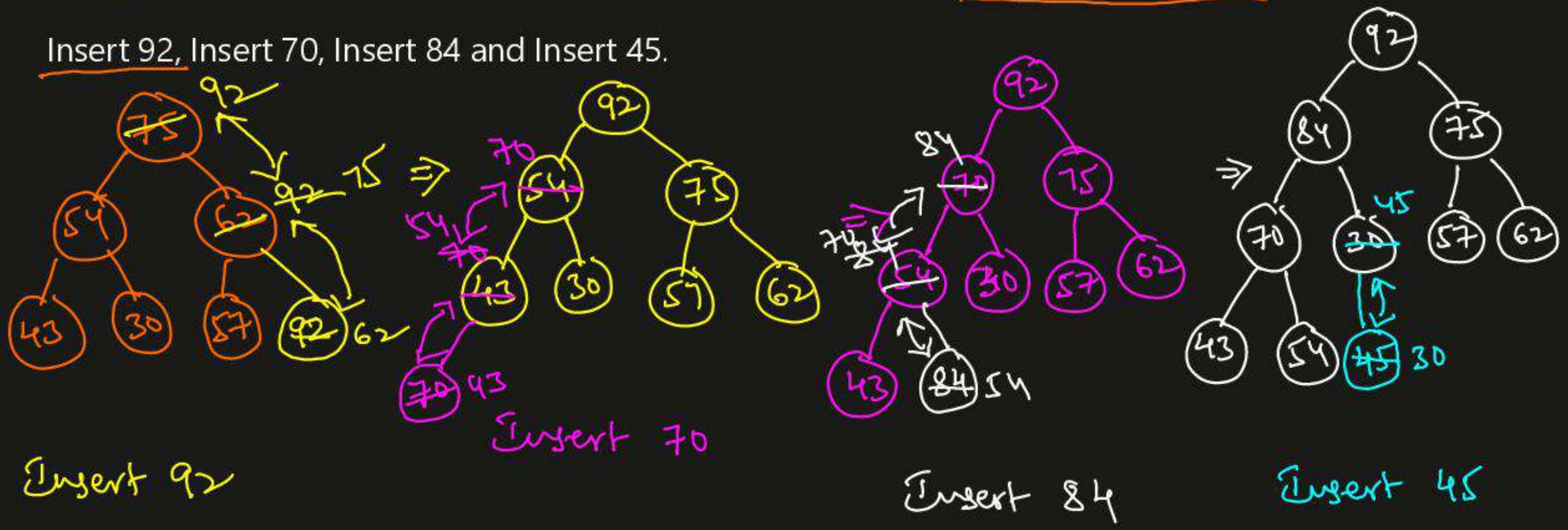
1. Insert the new item at the end of the heap.
2. Compare the newly inserted item with its parent.
3. If the parent is larger, stop. If the parent is smaller, swap the item with its parent.
4. Repeat step 2 until the parent is larger or equal to the item.

BYJU'S EXAM PREP

Insertion into Max-Heap

Result Max Heap: 92, 84, 75, 70, 45, 57, 62, 43, 54, 30

Example: Assume a Max heap with elements in the level order 75, 54, 62, 43, 30, 57. Now, Insert 92, Insert 70, Insert 84 and Insert 45.



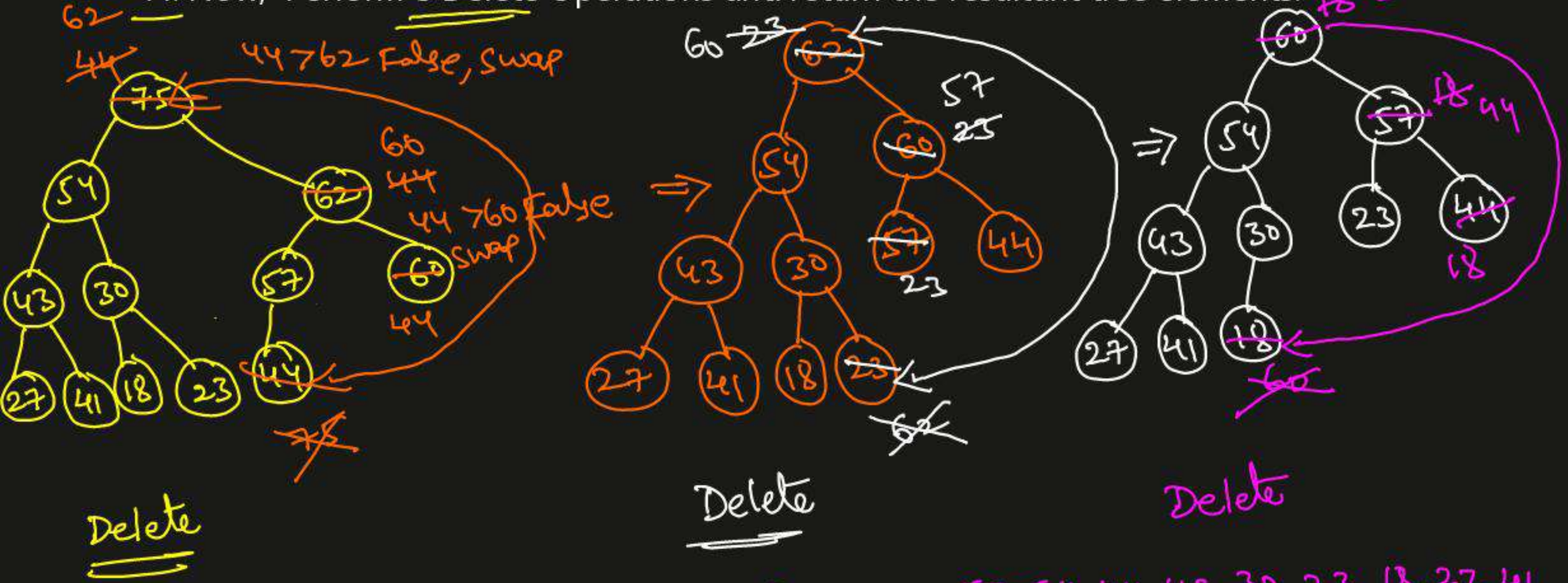
By default at "Root node"

Deletion / Find Max from Max - Heap

1. Store the first time in the tree in some temporary variable.
2. Replace the first node with the item in the last node.
3. Check the first node with its children nodes.
    - If the left child is larger, we swap it with the left child.
    - If the right node is larger, we swap it with the right node.
4. Repeat step 3 until the parent node is larger than the left and right child node.
5. Return the maximum value (stored in the temporary variable).

Binary Heap

Q1. Assume a Max heap with elements in the level order 75, 54, 62, 43, 30, 57, 60, 27, 41, 18, 23, 44. Now, Perform 3 Delete Operations and return the resultant tree elements.



Result : 57, 54, 44, 43, 30, 23, 18, 27, 41

Binary Heap

Q2. A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is given below :

Home-work

10, 8, 5, 3, 2

Two new elements "1' and "7' are inserted in the heap in that order. The level-order traversal of the heap after the insertion of the elements is :

[GATE 2005]

A. 10, 8, 7, 5, 3, 2, 1

B. 10, 8, 7, 2, 3, 1, 5

C. 10, 8, 7, 1, 2, 3, 5

D. 10, 8, 7, 3, 2, 1, 5

Binary Heap

Home-work

Q3. Which of the following sequences of array elements forms a heap ?

A {23, 17, 14, 6, 13, 10, 1, 12, 7, 5}

B {23, 17, 14, 6, 13, 10, 1, 5, 7, 12}

C {23, 17, 14, 7, 13, 10, 1, 5, 6, 12}

D {23, 17, 14, 7, 13, 10, 1, 12, 5, 7}

[GATE 2006]

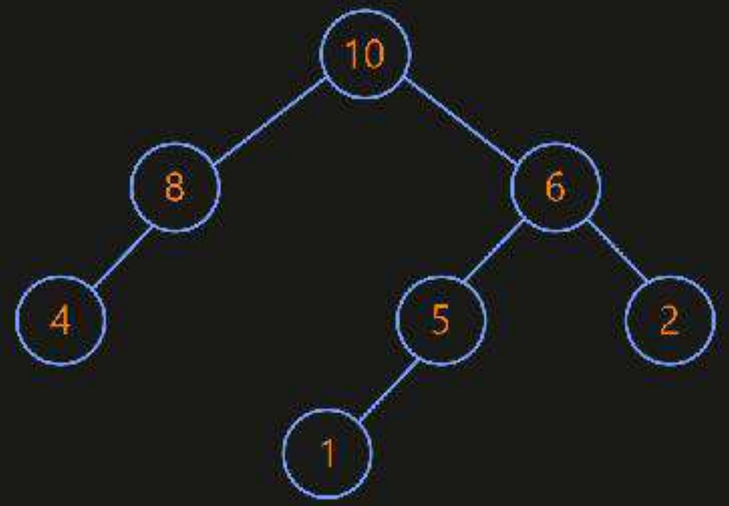
Binary Heap

Home-work

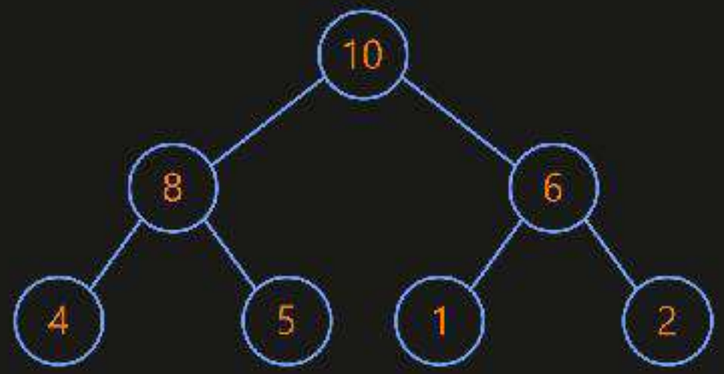
Q4. A max-heap is a heap where the value of each parent is greater than or equal to the val children. Which of the following is a max-heap ?

[GATE 2011]

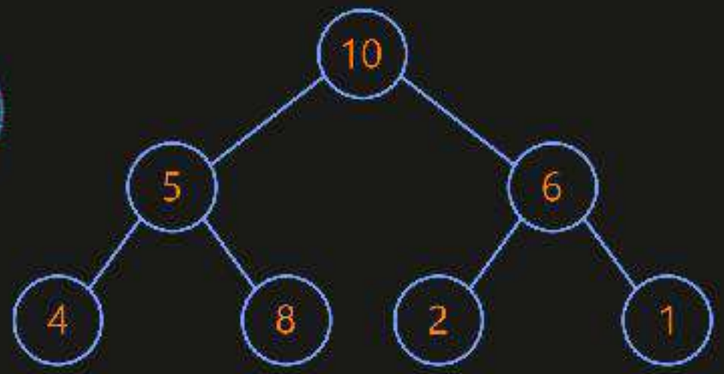
A



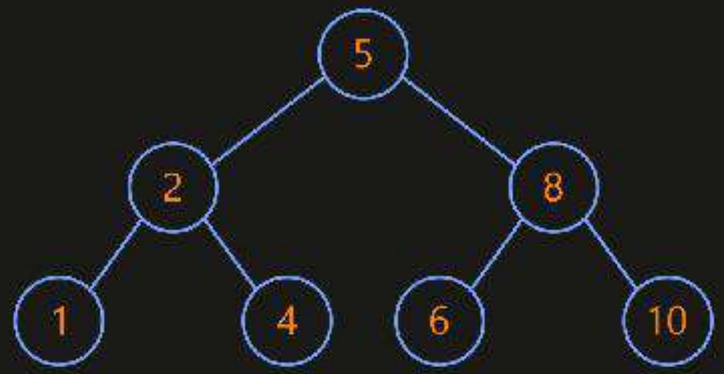
B



C



D

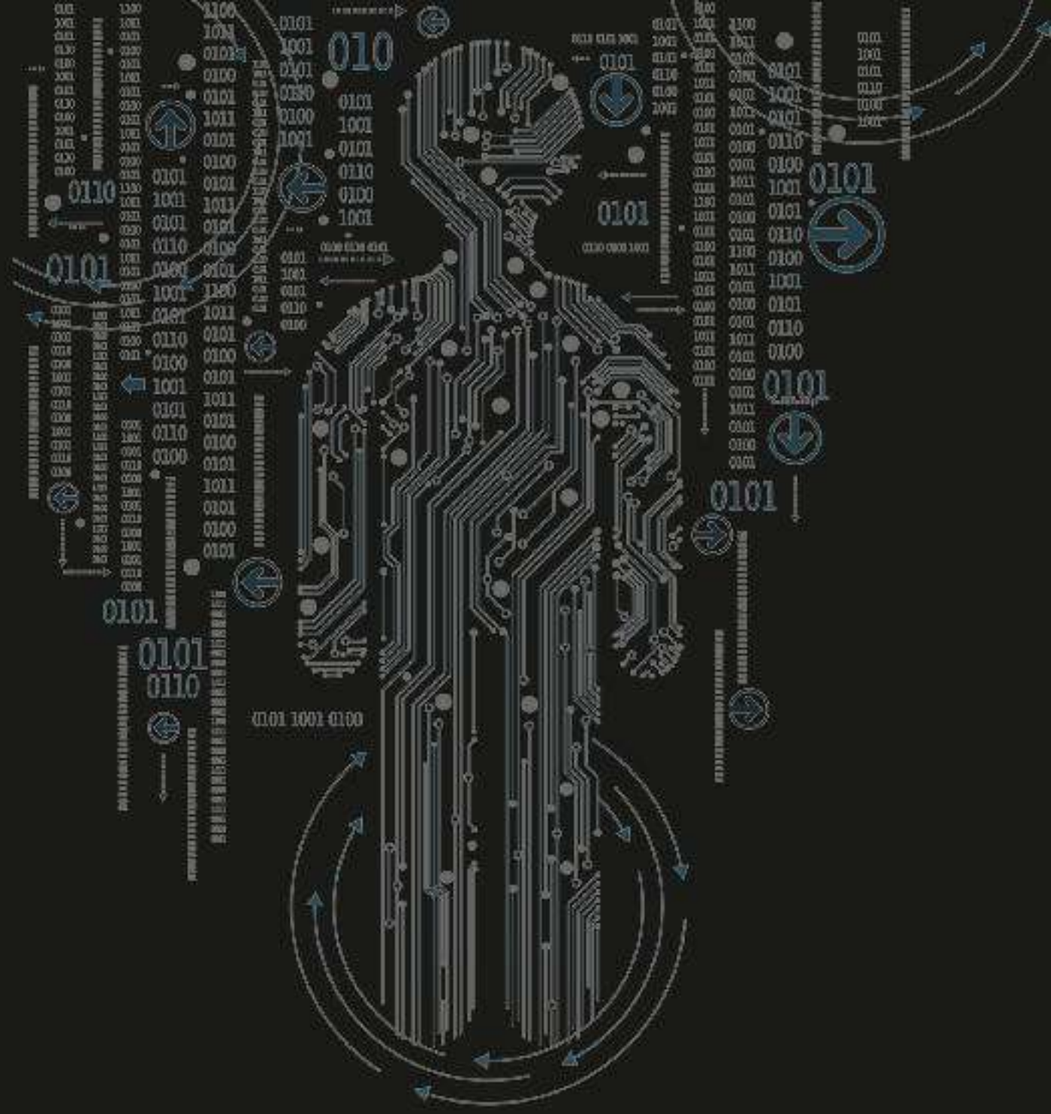


BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

AVL Trees - 1



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

AVL Trees - 1

BYJU'S EXAM PREP

Binary Search Tree - Insertion

BST: Left Subtree < Parent < Right Subtree

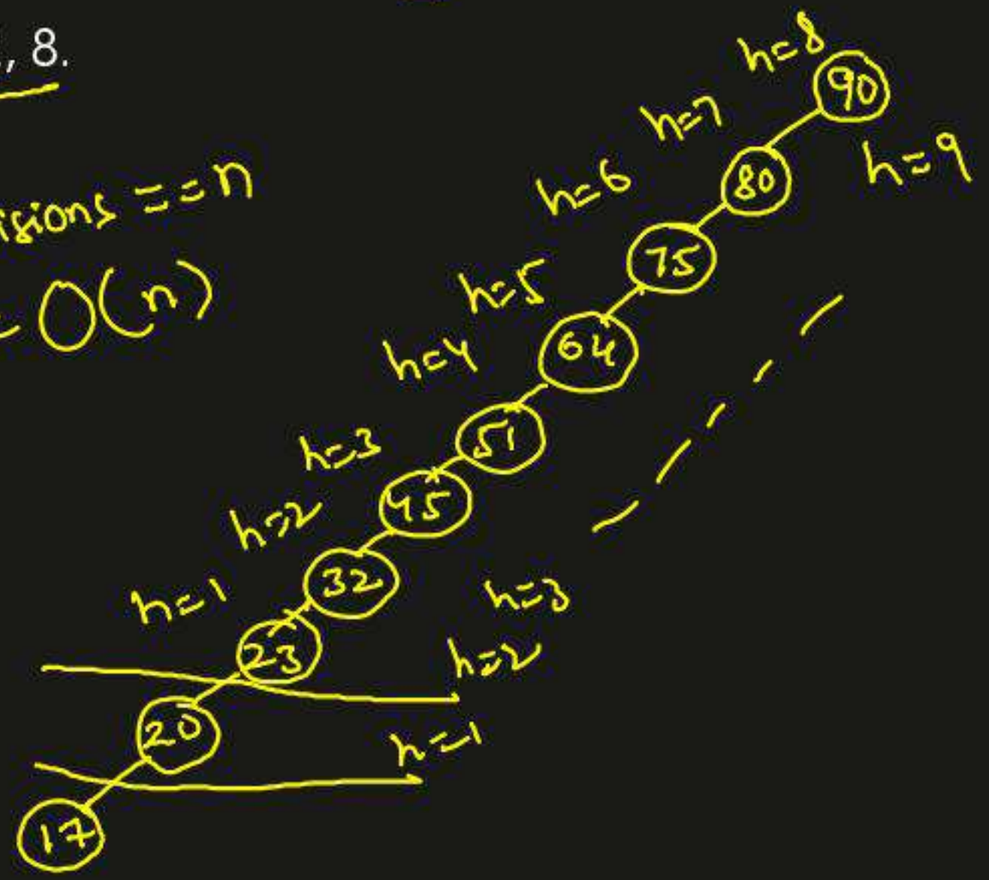
height number from '1'

Consider a BST with elements 90, 80, 75, 64, 51, 45, 32, 23. Now, Perform Insert Operations with elements 20, 17, 12, 8.

Number of Comparisions == n

Time Complexity = O(n)

* No-of Comparisions ∝ height of BST



80 < 90

75 < 90, 75 < 80

Insert 20: 20<90, 20<80, 20<75, 20<64, 20<51, 20<45, 20<32, 20<23

Insert 17: 17<90, 17<80, 17<75, ---- 17<20

BYJU'S EXAM PREP

Binary Search Tree - Insertion

height numbering from '1'

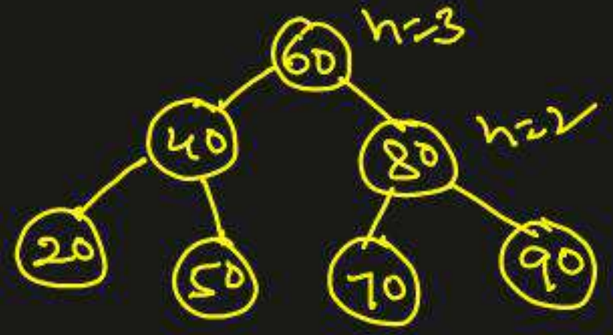
The Time Complexity of BST for insertion operation or search operation is O(n) in the worst case (Left Skewed / Right Skewed Binary Tree).

Consider BST with elements 60, 40, 80, 20, 50, 70, 90

No of Nodes = 7

No of comparisions = 3

$\lceil \log_2 7 \rceil == 3$



height = 3

Insert 10 : 10<60, 10<40, 10<20 No of Comparisions = 3

Insert 55 : 55<60, 55>40, 55<50 ⇒ " = 3

Insert 82 : 82>60, 82>80, 82<90 ⇒ " = 3

No of Comparisions ∝ height of BST

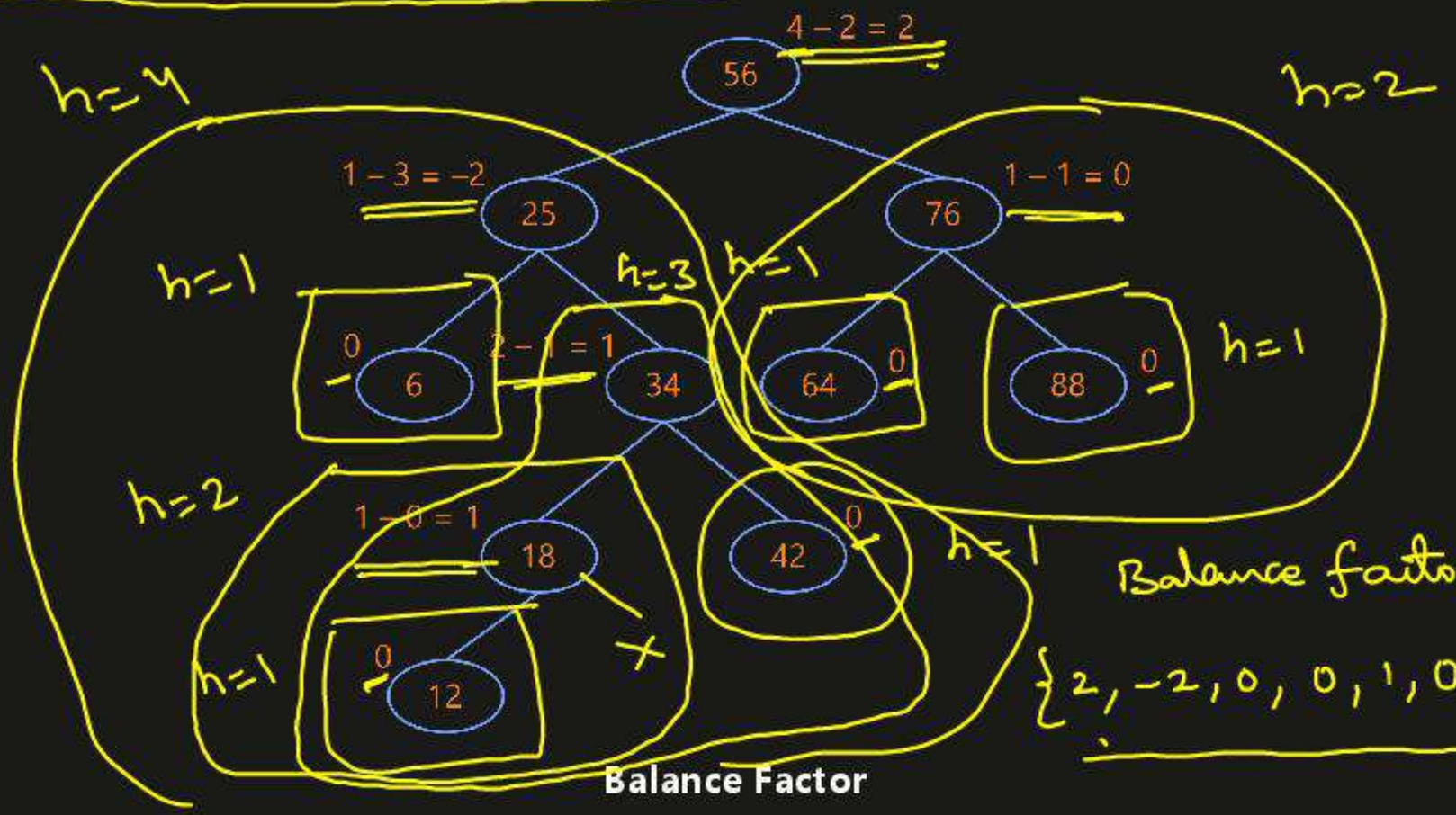
O(n) ⇒ O(log n) always

BYJU'S EXAM PREP

Balance Factor

Balance factor of leaf Nodes == 0
height numbering from '1'

Balance Factor of a node = height of left subtree – height of right sub tree



Balance factors

{2, -2, 0, 0, 1, 0, 0, 1, 0, 0}

AVL Trees - 1

Balance factor = {-1, 0, 1}

AVL Tree

Balanced BST == AVL Tree

|height of left subtree − height of right subtree|

AVL tree is a binary search tree in which the difference of heights of left and right subtrees of any node is less than or equal to one. The technique of balancing the height of binary trees was developed by Adelson, Velskii, and Landi and hence given the short form as AVL tree or Balanced Binary Tree.

The Balance factor of a node in a binary tree can have value 1, -1, 0, depending on whether the height of its left subtree is greater, less than or equal to the height of the right subtree.

BYJU'S EXAM PREP

Rotations

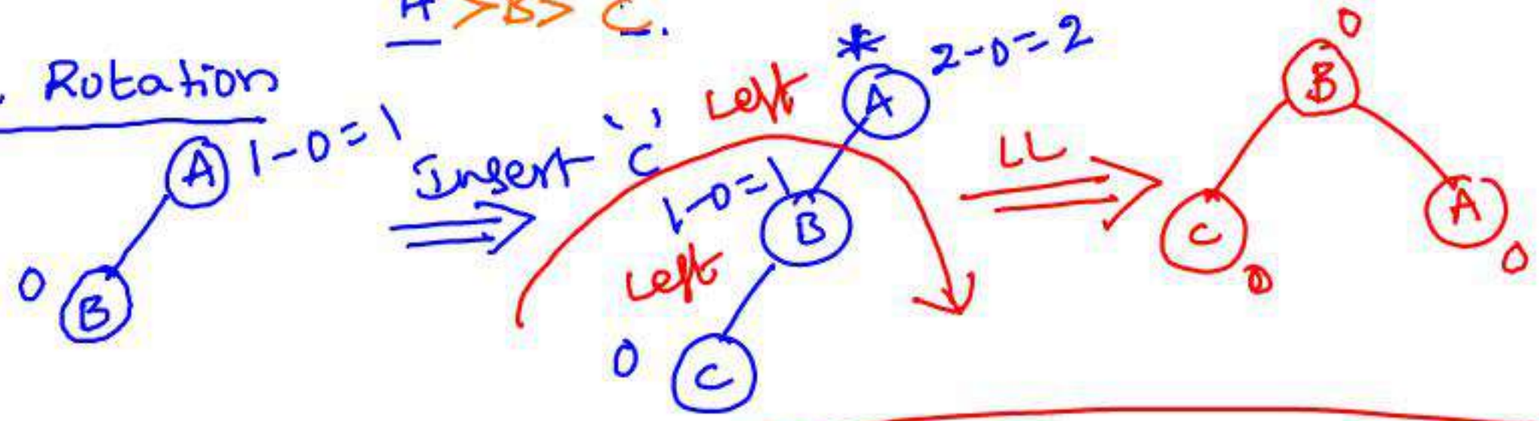
To Convert BST into AVL Tree (Balanced BST), we use rotations

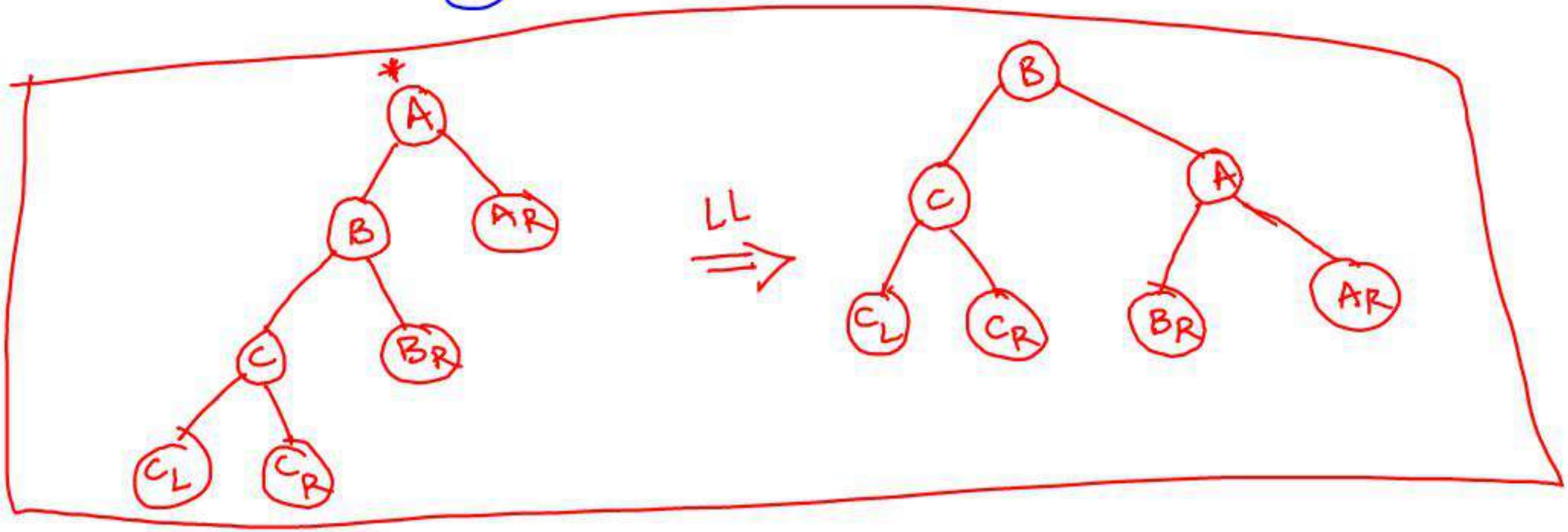
To balance a BST, either of 4 rotations can be performed.

1. L L rotation: Inserted node is in the left subtree of left subtree of A
2. R R rotation : Inserted node is in the right subtree of right subtree of A
3. L R rotation : Inserted node is in the right subtree of left subtree of A
4. R L rotation : Inserted node is in the left subtree of right subtree of A

LL Rotation

A > B > C.

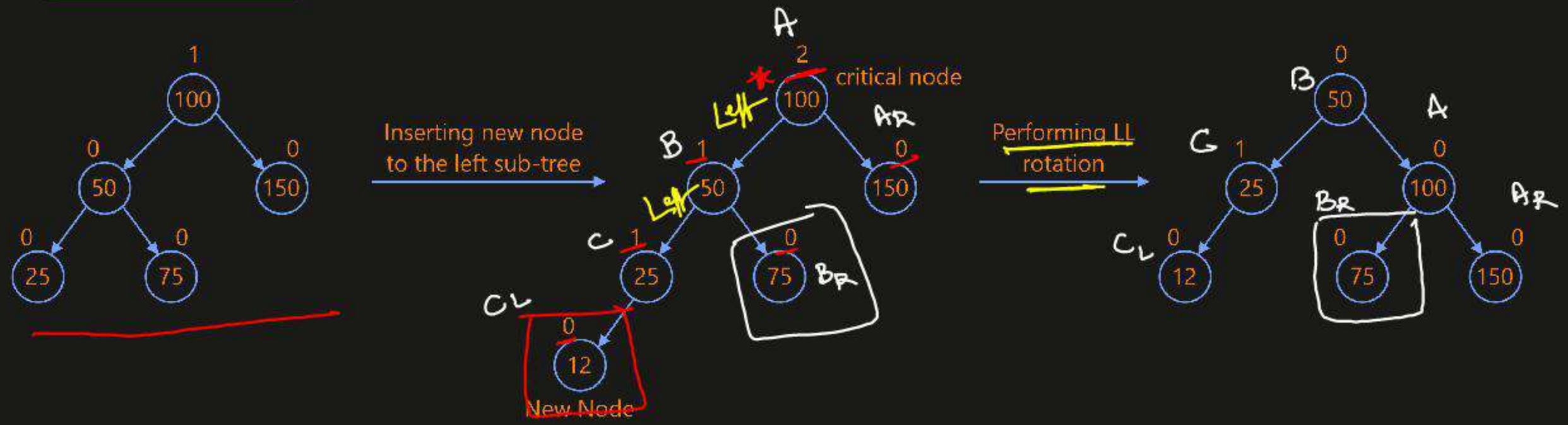




Node which is not balanced is called as "CRITICAL NODE *"

BYJU'S EXAM PREP

LL Rotation



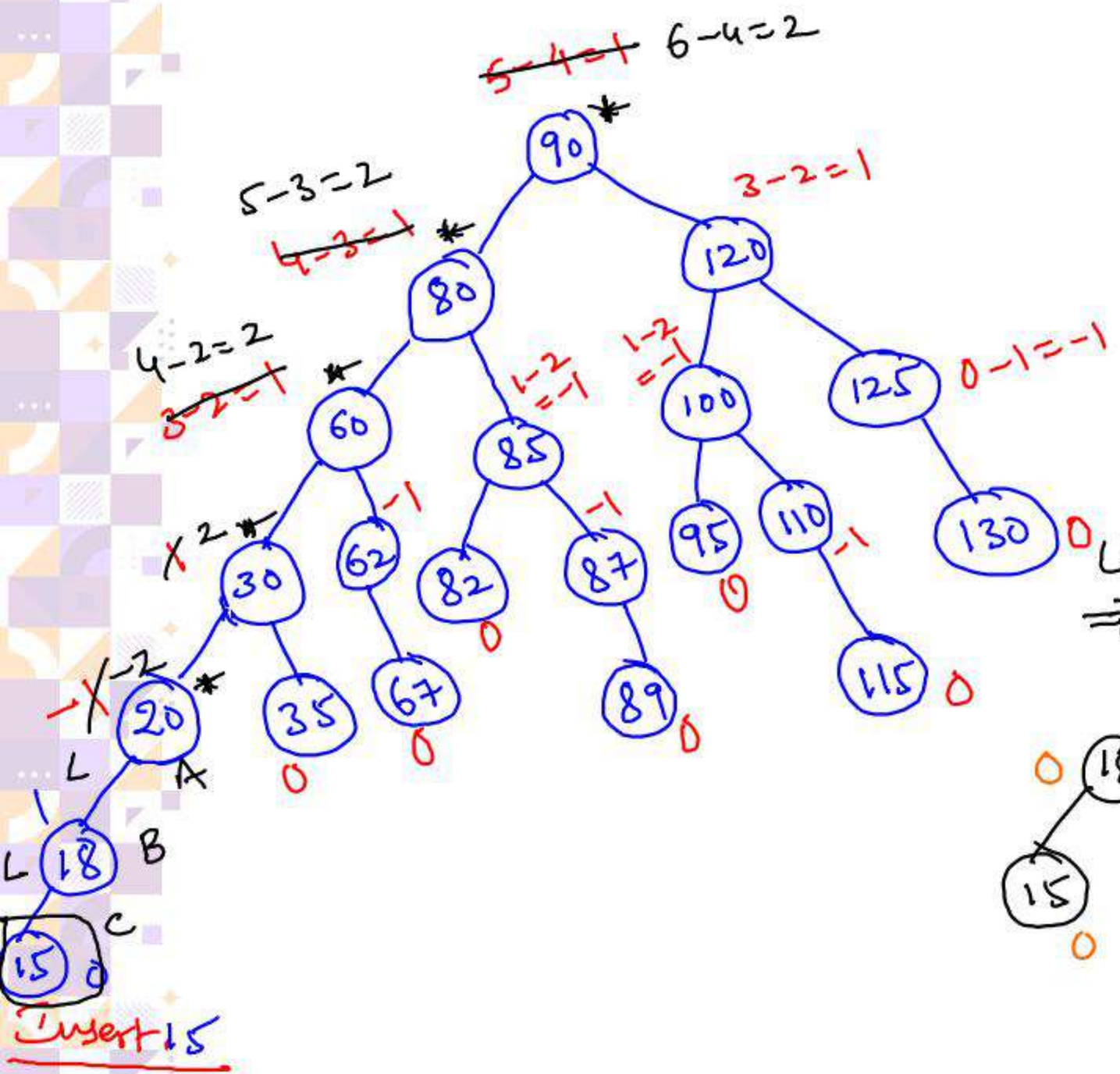
AVL Tree

Non-AVL Tree

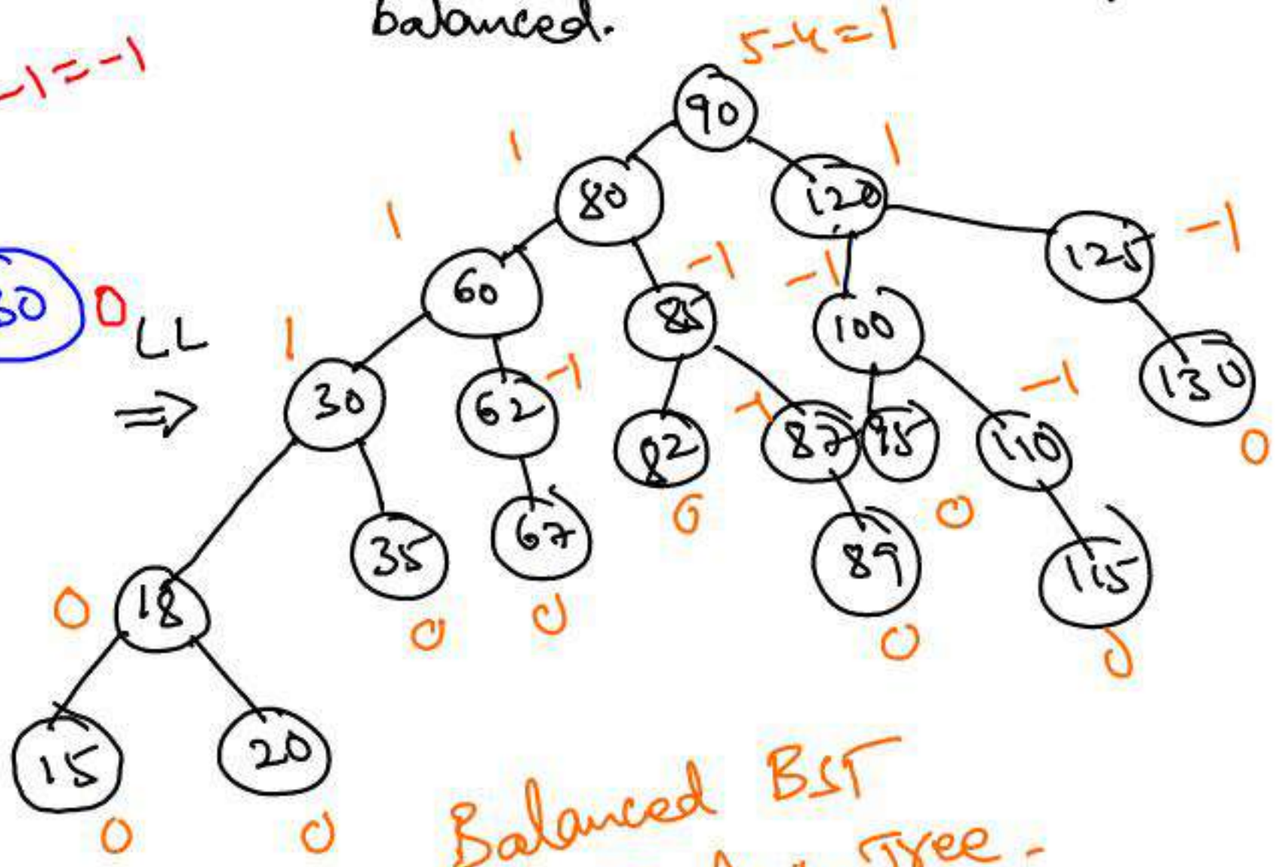
LL Rotated Tree

$\{-1, 0, +1\}$

NOTE: when multiple critical nodes, bottom most critical node is balanced, others automatically gets balanced.

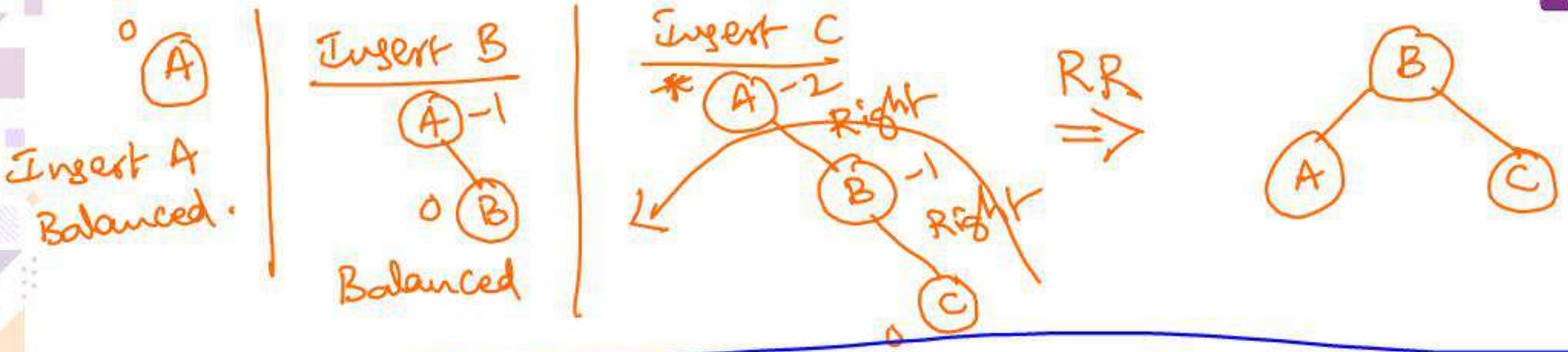


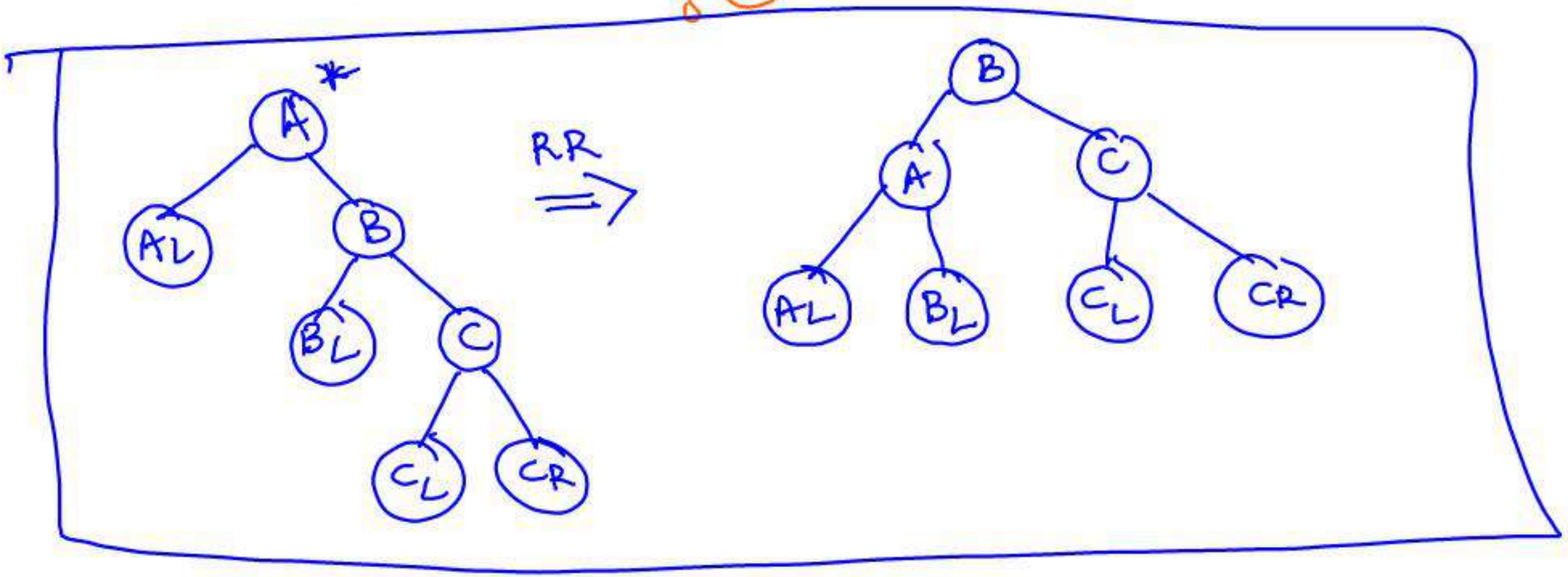
LL ⇒



Balanced BST
⇒ AVL Tree.

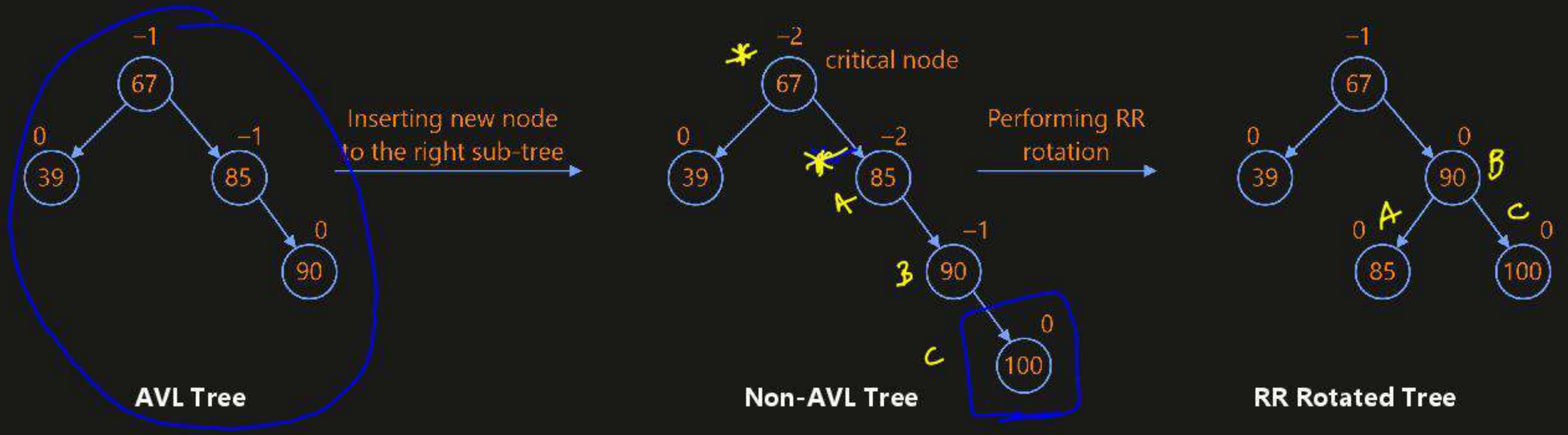
$A < B < C$


0
A
Insert A
Balanced.
Insert B
A -1
0 B
Balanced
Insert C
* A -2
Right
B -1
Right
C
0
RR
B
A
C


*
A
AL
B
BL
C
CL
CR
RR
B
A
C
AL
BL
CL
CR

BYJU'S EXAM PREP

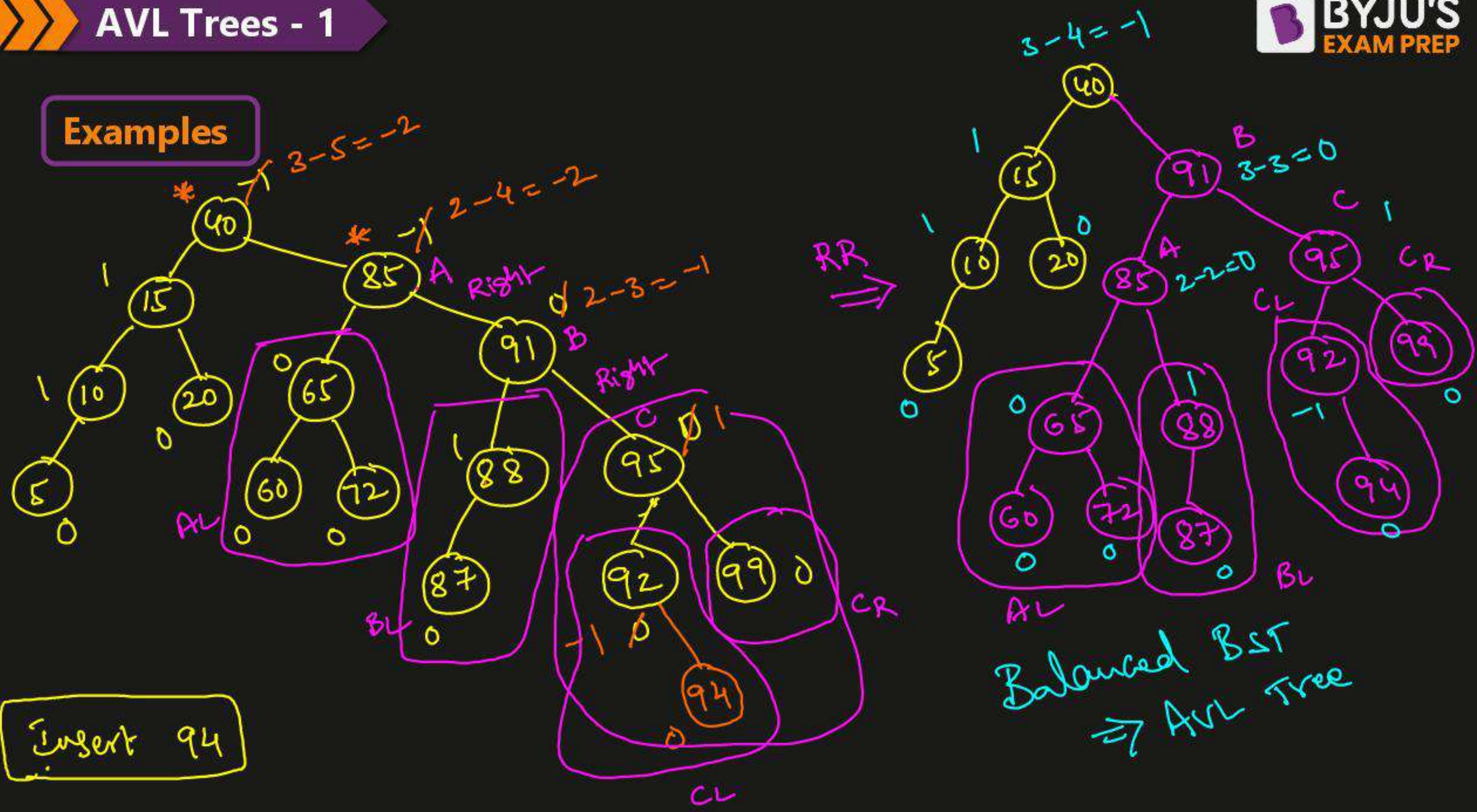
RR Rotation



AVL Trees - 1

BYJU'S EXAM PREP

Examples



- LR Rotation
- RL Rotation

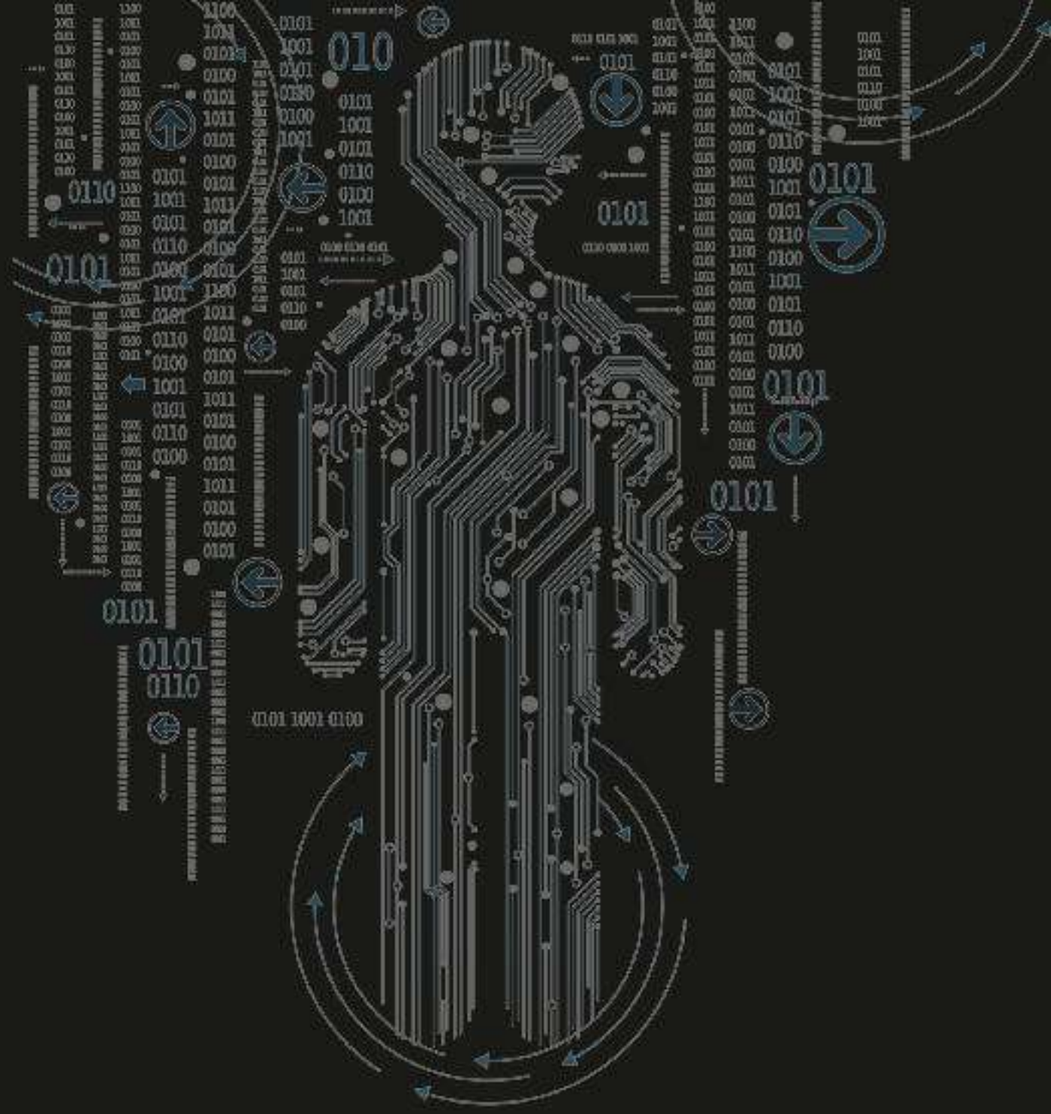
} To be contd ...

BYJU'S EXAM PREP
4.5 Google Play Review
GET IT ON Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

AVL Trees - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Last Class – Quick Revision

- Time complexity for Insertion | Search in BST $\propto$ height of BST

  Best case – $O(\log n)$ Average, worst case – $O(n)$

- To make T.C to $O(\log n)$ in all cases, height should be balanced.
- height balanced $\Rightarrow$ Balance factor = $\{-1, 0, +1\}$
- Balance factor = height of left subtree – height of right subtree
- If not balanced, Then to balance Rotations are Performed.

  ✓LL LR RL RR✓

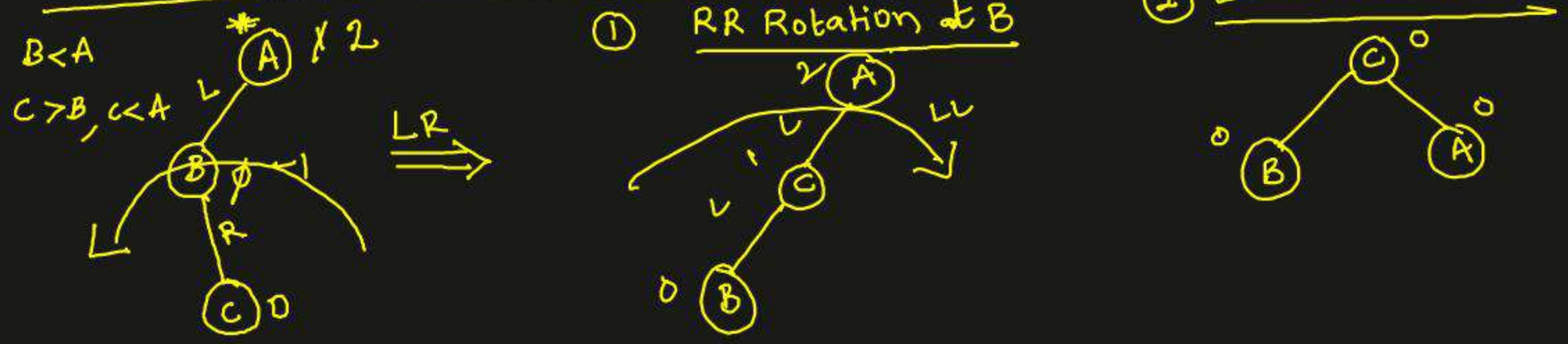
LR Rotation, RL Rotation called as Double Rotations.

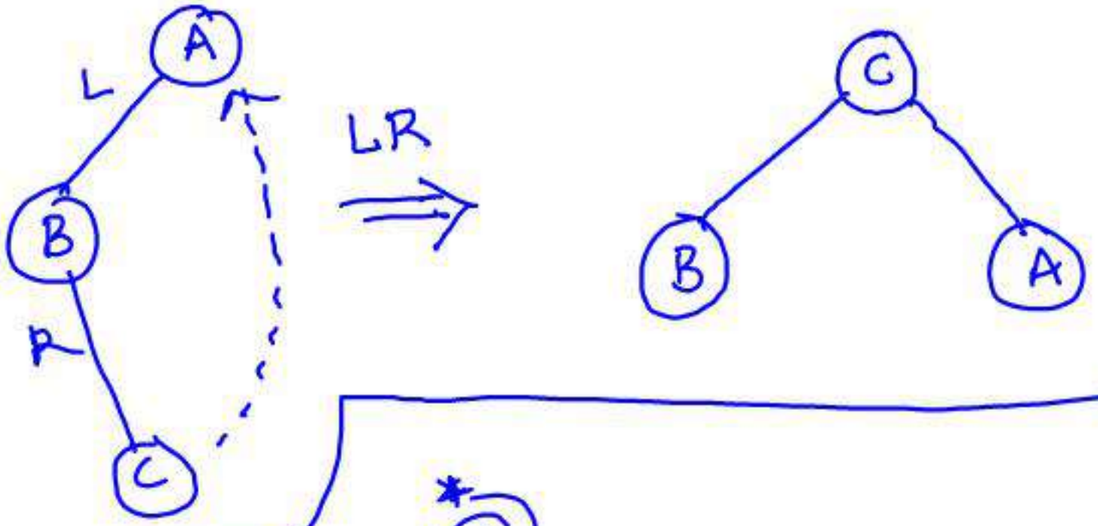
LR Rotation

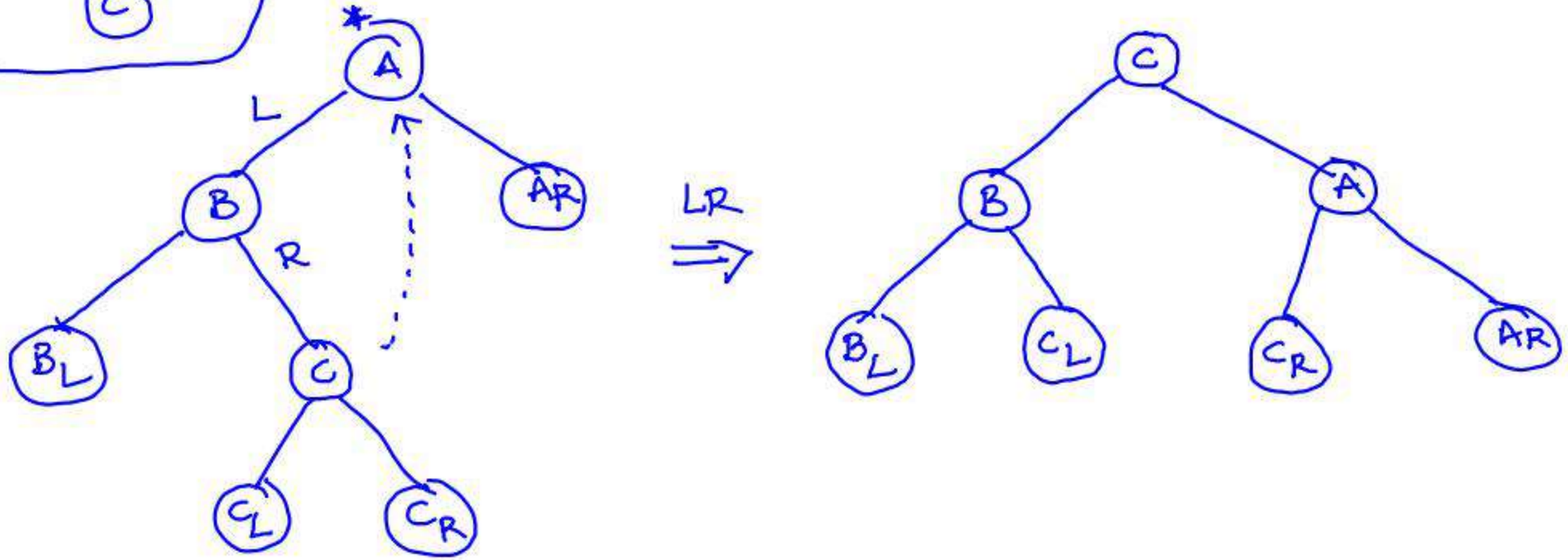
LR : RR + LL

RL : LL + RR

Double rotations are bit tougher than single rotation which has already explained above. LR rotation = RR rotation + LL rotation, i.e., first RR rotation is performed on subtree and then LL rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.

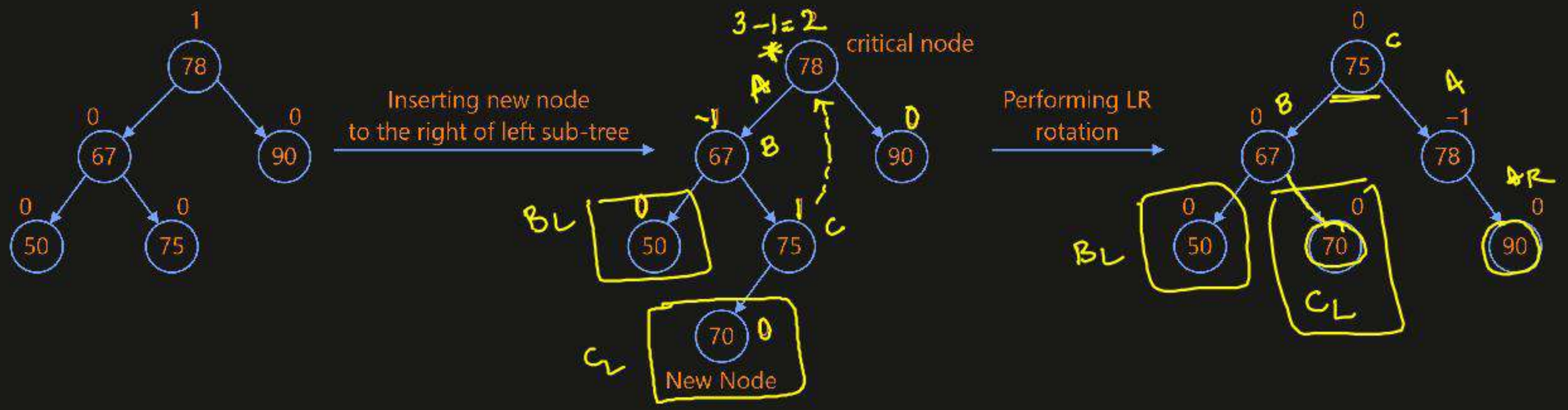



A
L
B
R
C
LR
C
B
A


*
A
L
B
AR
R
BL
C
CL
CR
LR
C
B
A
BL
CL
CR
AR

BYJU'S EXAM PREP

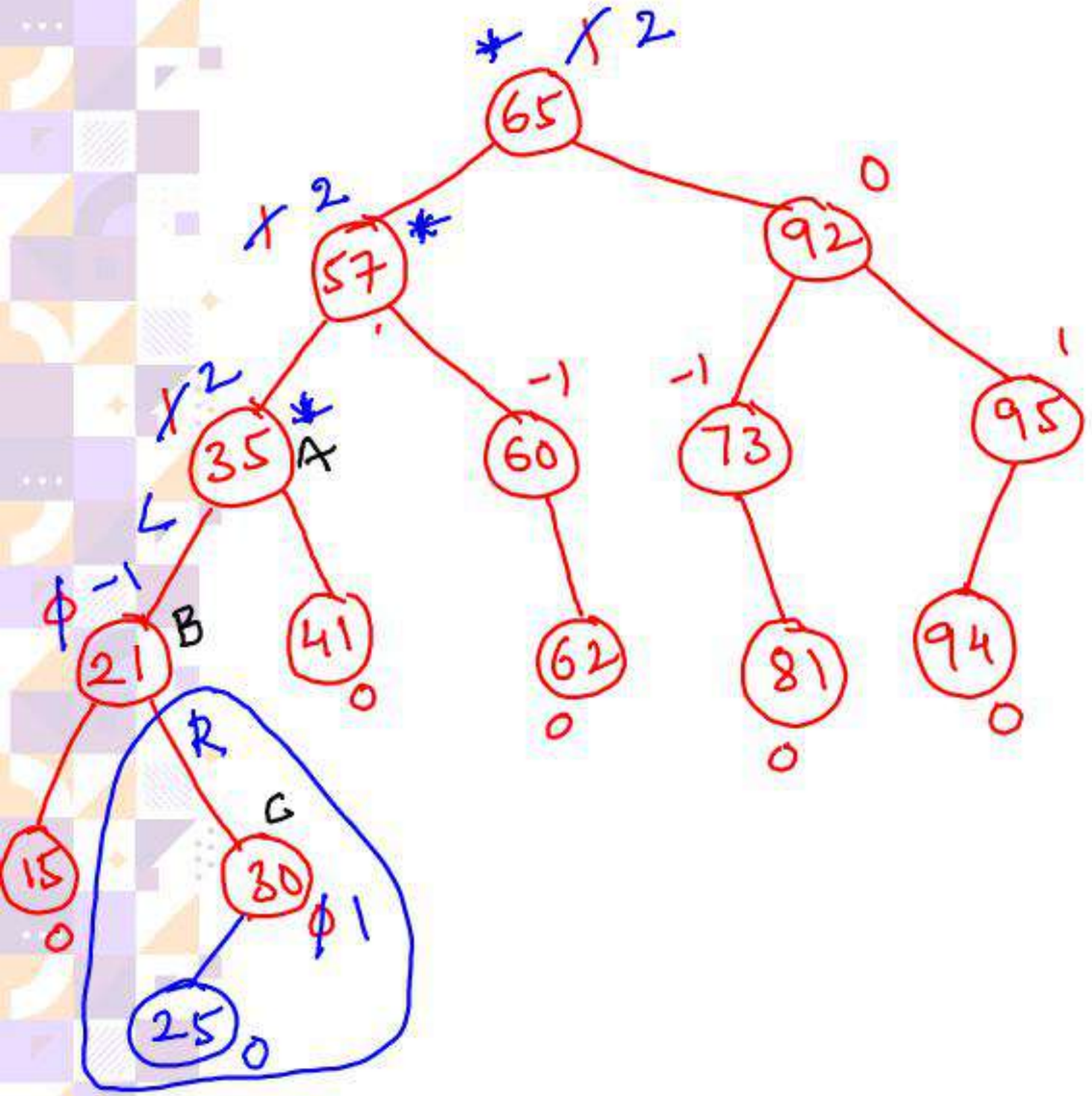
LR Rotation



AVL Tree

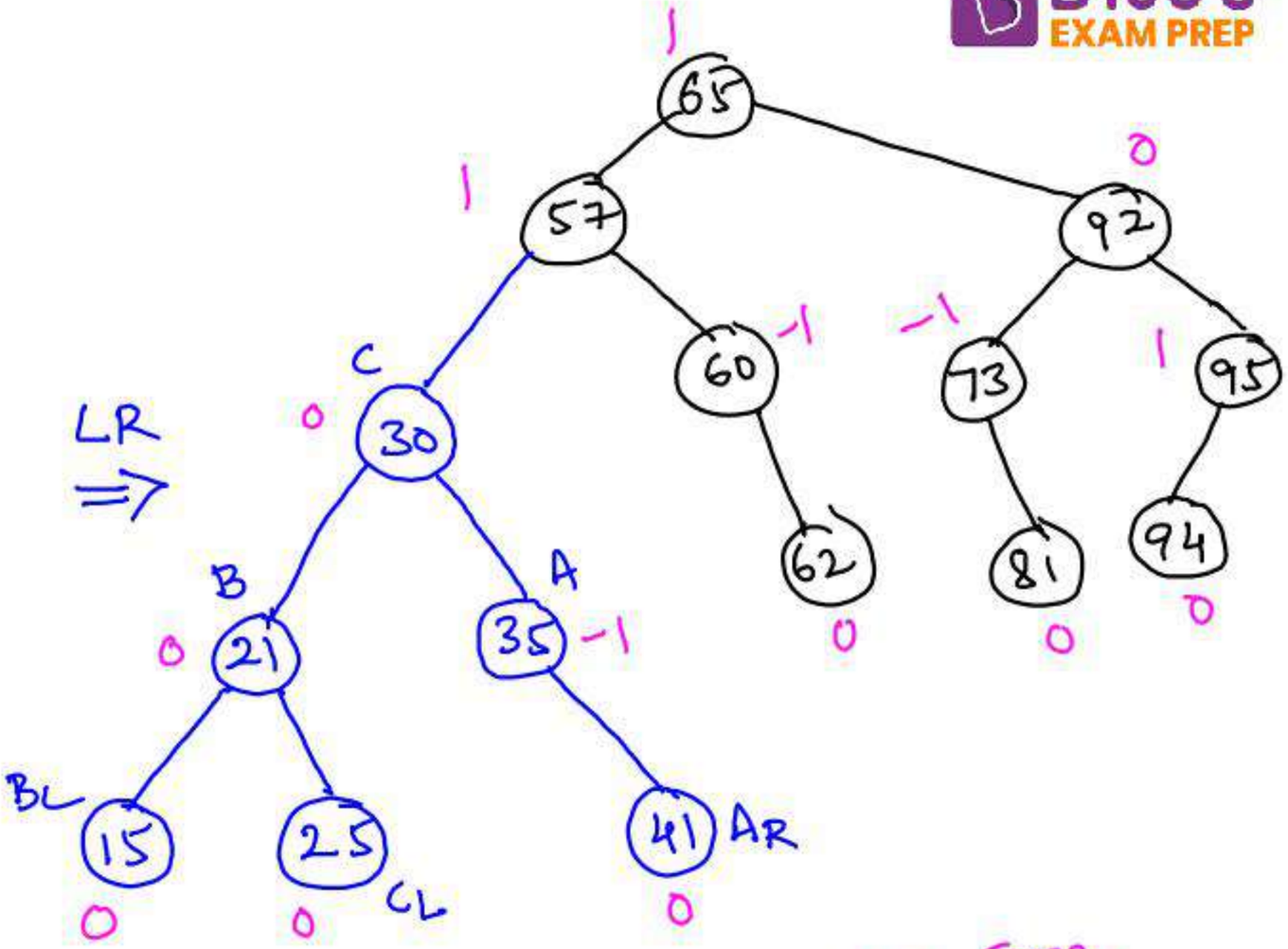
Non-AVL Tree

LR Rotated Tree



Insert 25

LR ⇒

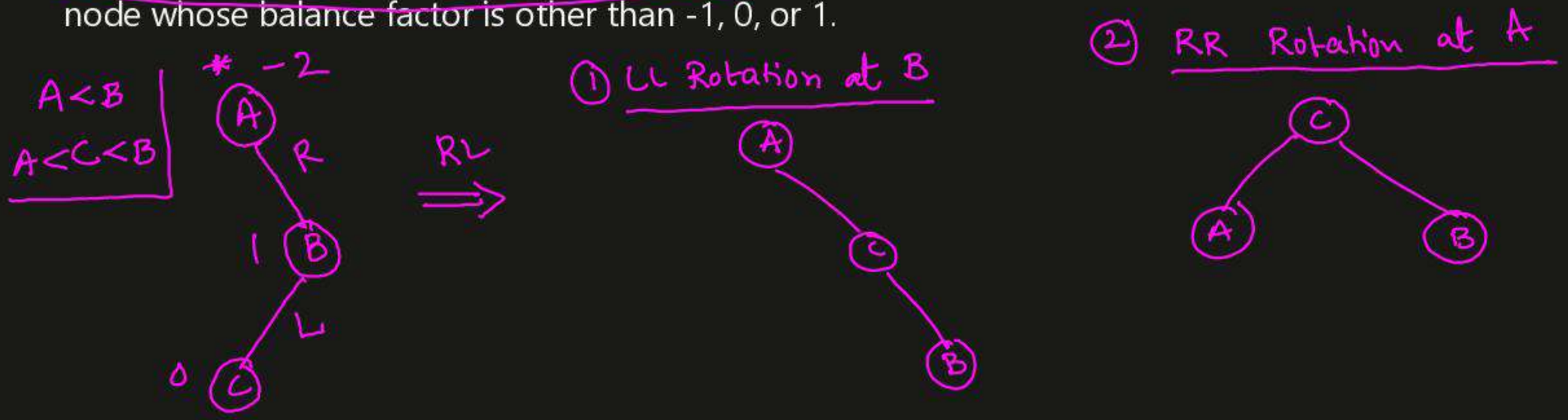


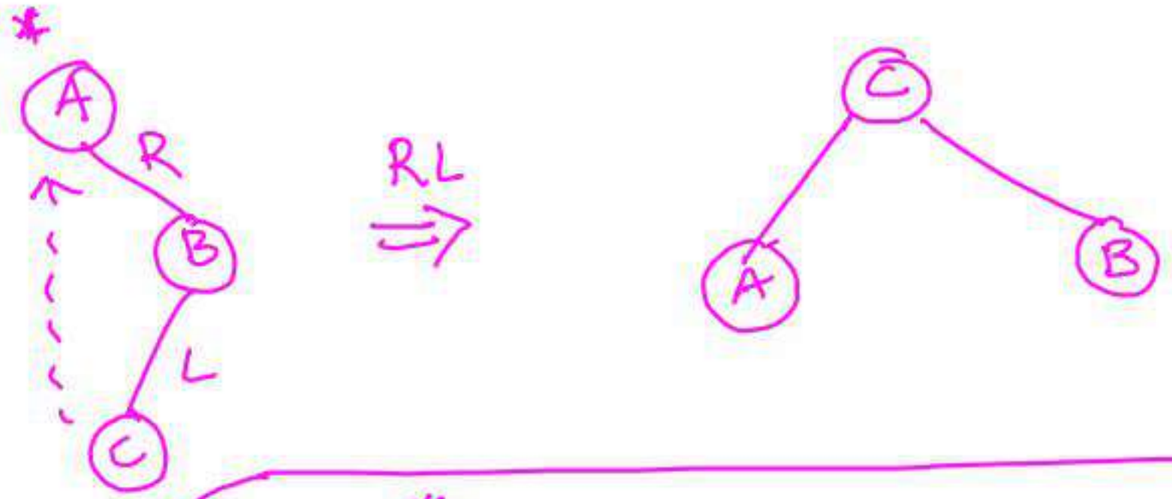
Balanced BST = AVL Tree

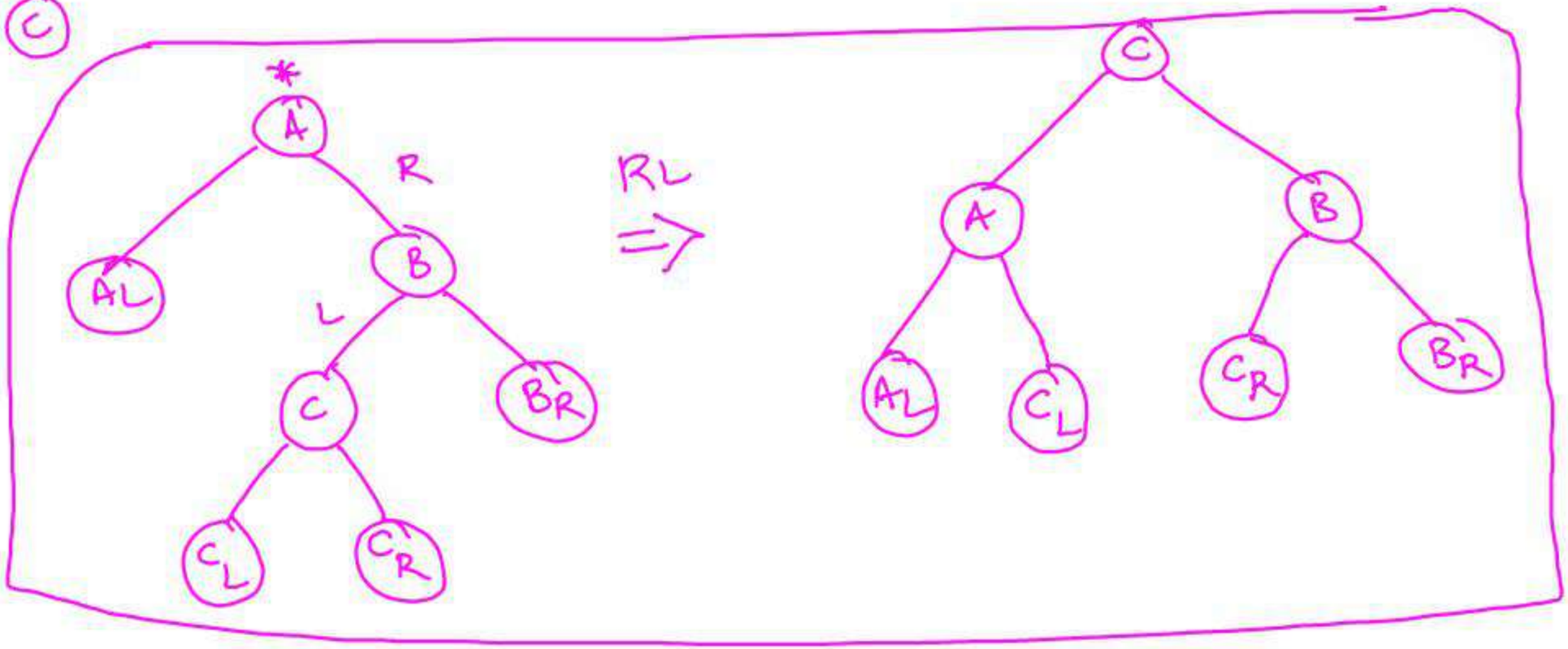
BYJU'S EXAM PREP

RL Rotation

RL rotation = LL rotation + RR rotation, i.e., first LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.

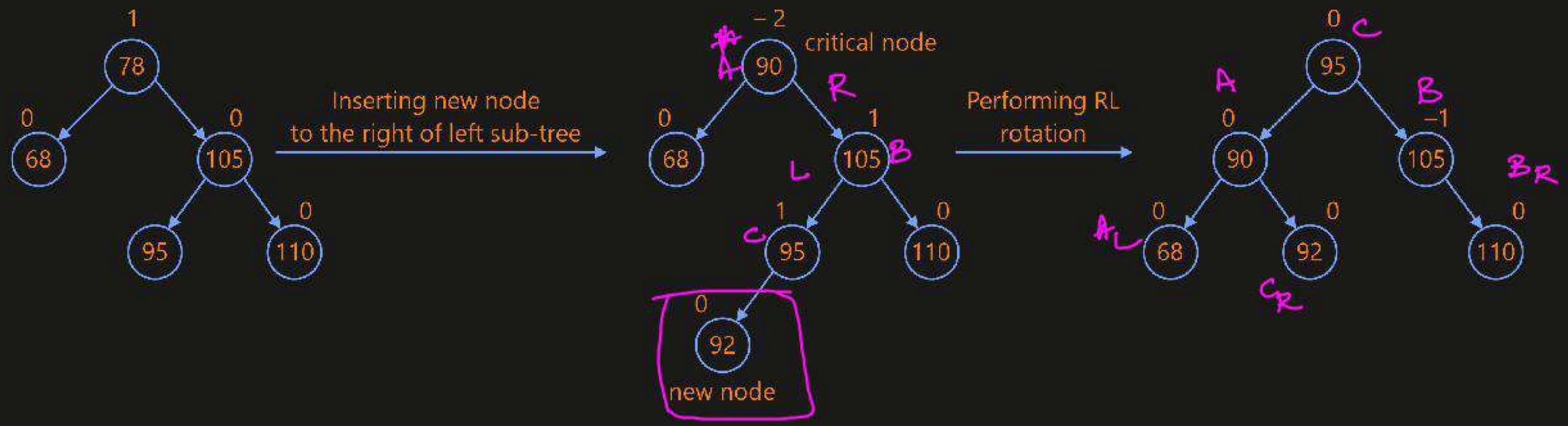



*
A
R
B
L
C
RL
C
A
B


*
A
R
AL
B
L
C
BR
CL
CR
RL
C
A
B
AL
CL
CR
BR

BYJU'S EXAM PREP

RL Rotation



AVL Tree

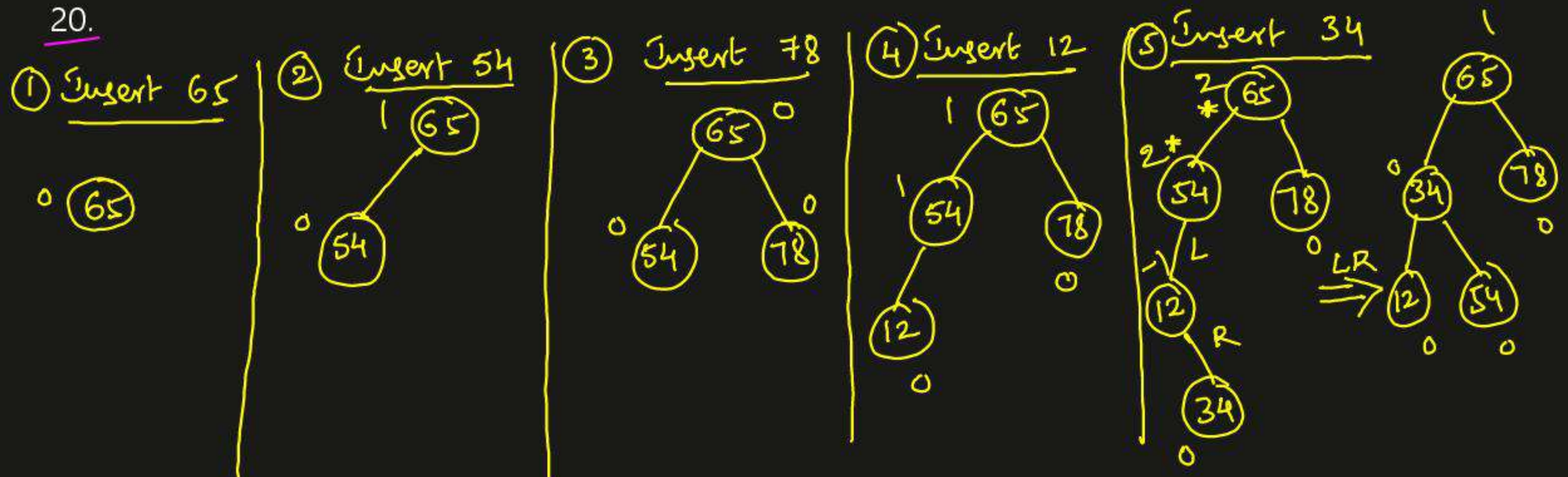
Non-AVL Tree

RL Rotated Tree

BYJU'S EXAM PREP

AVL Tree Construction

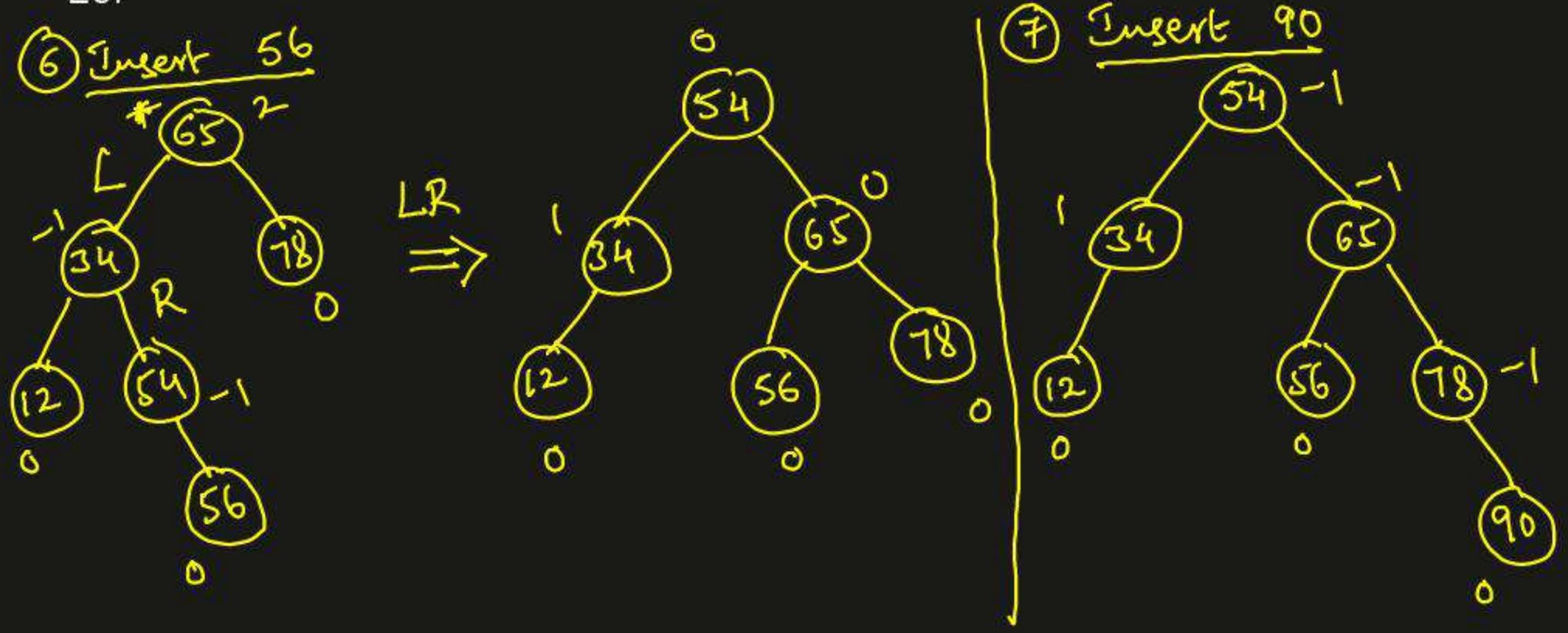
Construct an AVL Tree with elements inserted in the order 65, 54, 78, 12, 34, 56, 90, 19, 45, 60 and 20.



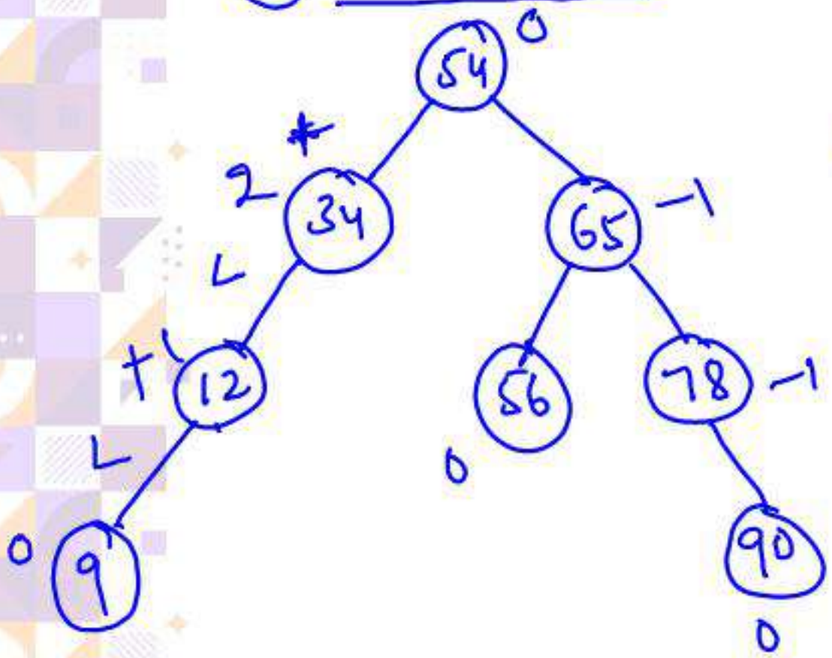
BYJU'S EXAM PREP

AVL Tree Construction

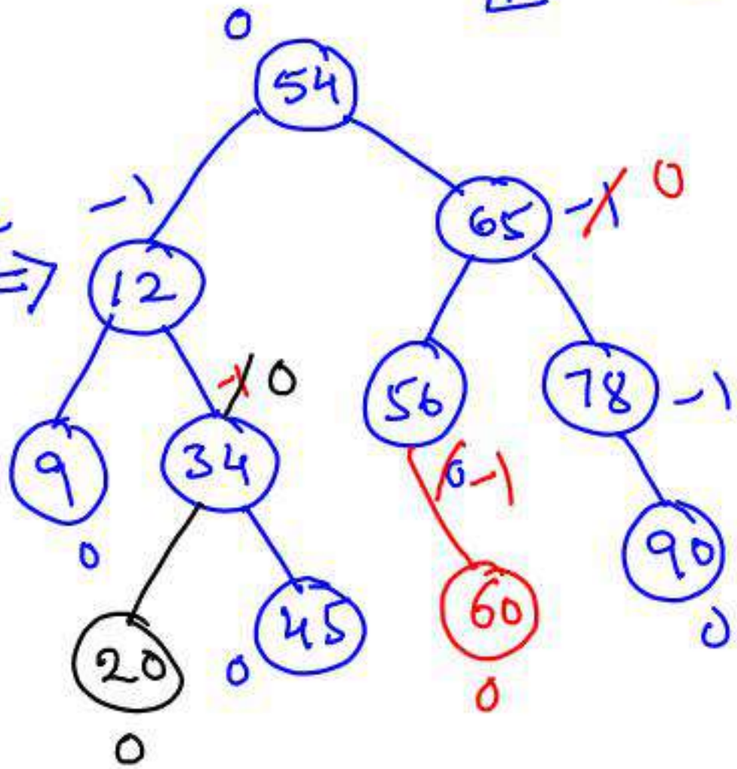
Construct an AVL Tree with elements inserted in the order 65, 54, 78, 12, 34, 56, 90, ~~10~~ 9, 45, 60 and 20.



(8) Insert 9



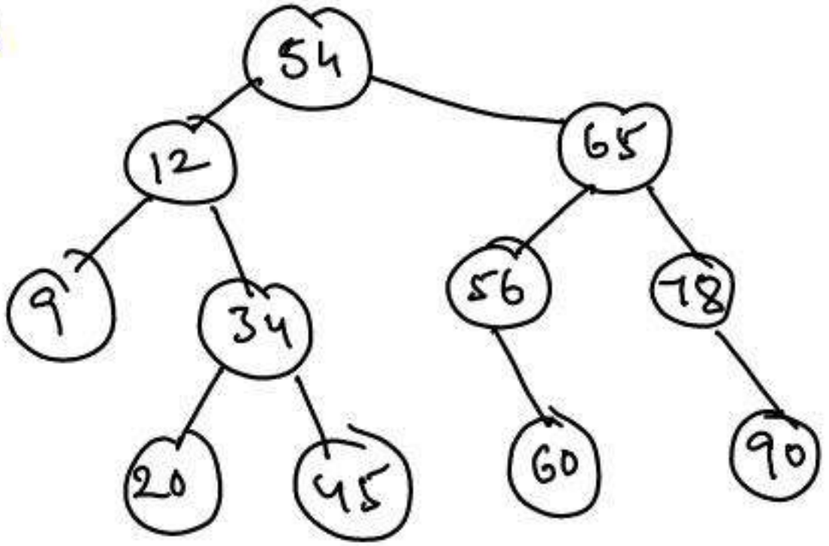
LL ⟹



(9) Insert 45

(10) Insert 60

(11) Insert 20



Balanced BST = AVL Tree.

BYJU'S EXAM PREP

AVL Tree Complexity

Algorithm	Average case	Worst case
Space	o(n)	o(n)
Search	o(log n)	o(log n)
Insert	o(log n)	o(log n)
Delete	o(log n)	o(log n)

AVL Trees - 2

unlabelled,

Labelled

Tree Formulae

1. The No. of Unlabelled binary trees Possible with 'n' nodes $= \dfrac{{}^{2n}C_n}{n+1}$

2. The No. of Labelled binary trees, Possible with 'n' nodes $= \dfrac{{}^{2n}C_n}{(n+1)} * n!$

3. Balance Factor = height of left subtree − height of right subtree
   [* height numbering from '1']

4. If there are 'l' leaf nodes, then the No. of Nodes with two children $= (l-1)$

BYJU'S EXAM PREP

Tree Formulae

(5) If 'N' nodes, the Maximum height $= (N-1)$

minimum height $= \lceil \log_2(N+1) \rceil - 1$

(6) For a tree with height, H, minimum No. of Nodes $= (H+1)$

Maximum No. of Nodes $= 2^{(H+1)} - 1$

(7) The Number of Nodes at level 'L' in PBT $= 2^L$

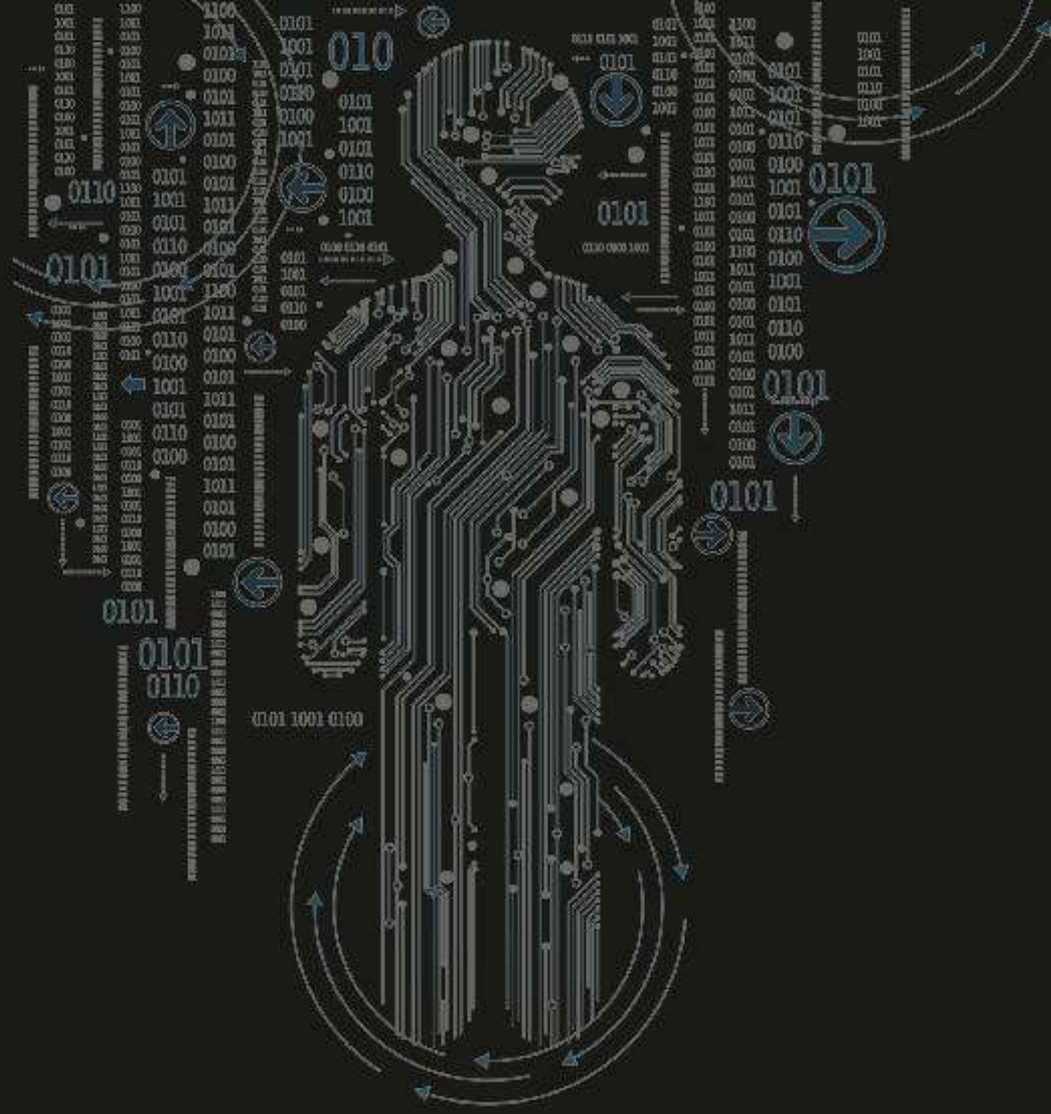
@ 6 PM – Practice session on DS

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Problem Solving @
Data Structures



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

BYJU'S EXAM PREP

Q1. The resultant elements after following operations In the same sequence, into a Binary Heap with elements,

10, 20, 15, 35, 32, 54, 48, 39

Insert 12, Insert 24, Delete, Delete

A. 15, 20, 35, 24, 32, 54, 48, 39

B. 15, 20, 35, 32, 24, 48, 54, 39

C. 15, 20, 35, 32, 54, 24, 48, 39

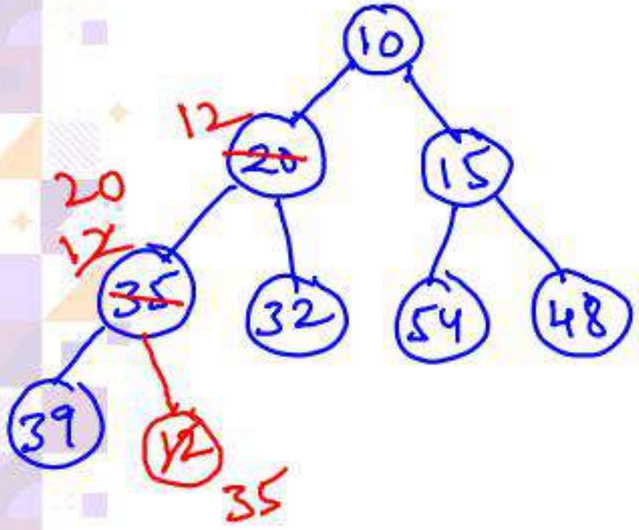
D. 15, 20, 35, 32, 24, 54, 48, 39

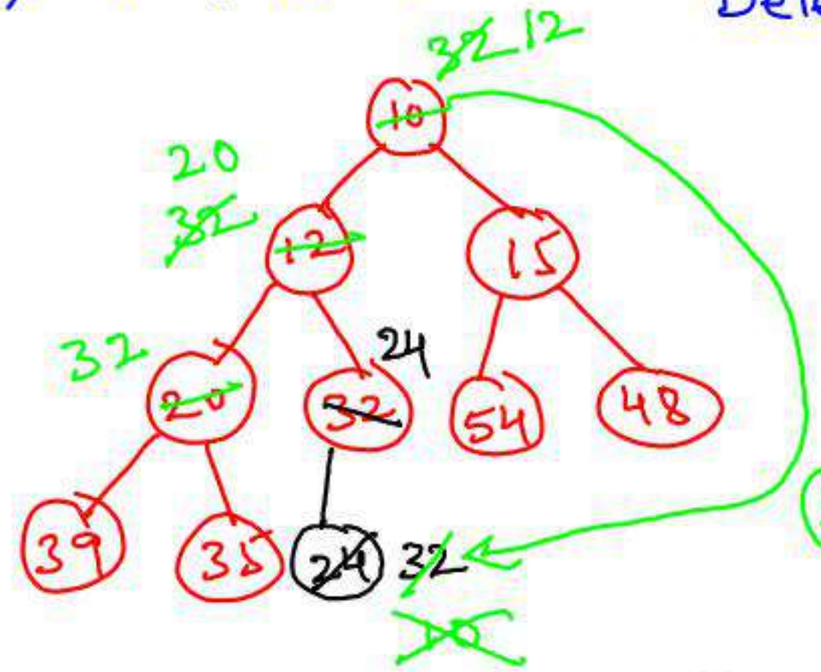
15, 20, 35, 32, 24, 54, 48, 39

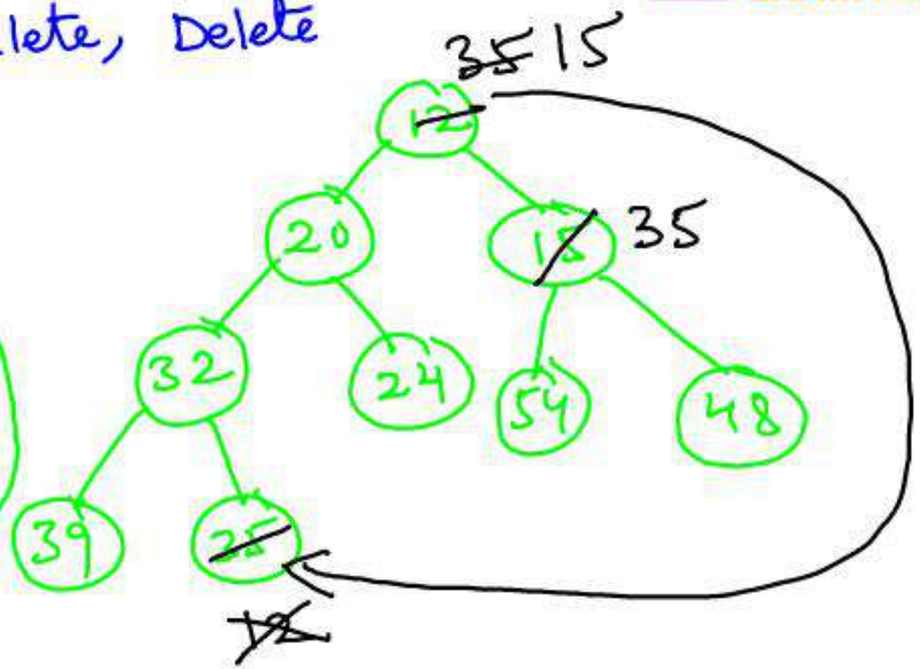
10, 20, 15, 35, 32, 54, 48, 39

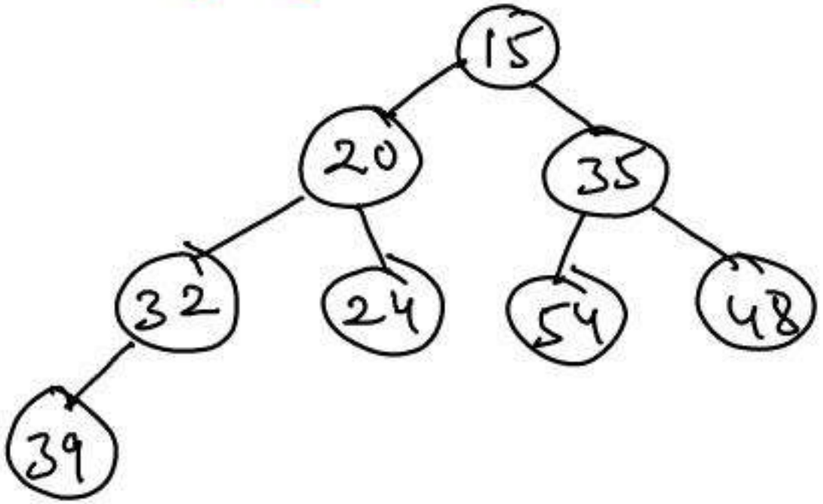
Insert 12, 24

Delete, Delete









BYJU'S EXAM PREP

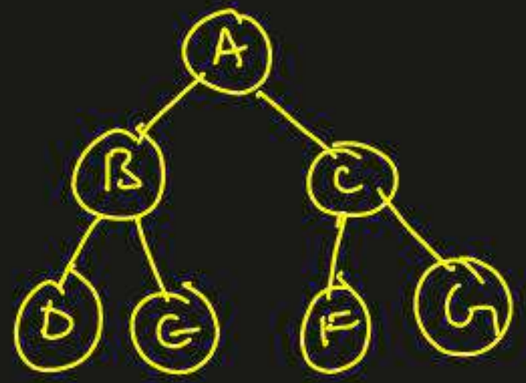
Q2. In a min heap, find worst case time complexity for finding maximum node by using optimal algorithm.

A. θ (1)

B. θ (log n)

C. θ (n)

D. θ (n log n)



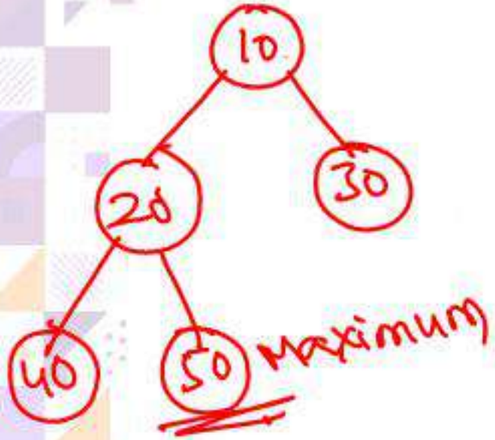
Total 7 Nodes

⇒ No. of leaf Nodes = 4

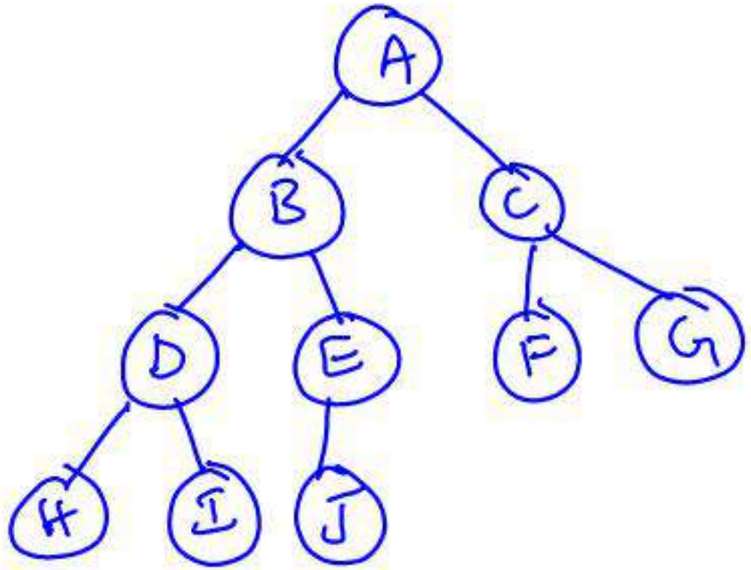
In Worst-case, Maximum Node present at leaf level.

'n' Nodes ⇒ Max No. of leaf Nodes in CBT ⇒ $\left\lceil \frac{n}{2} \right\rceil$

⇒ No. of Comparisions = $O\left(\frac{n}{2}\right)$ ⇒ $O(n)$



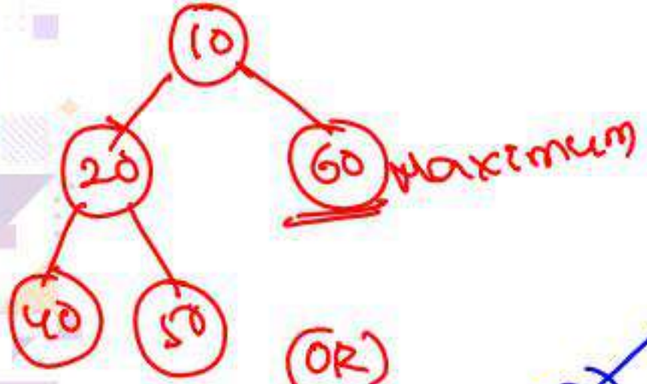
(OR)



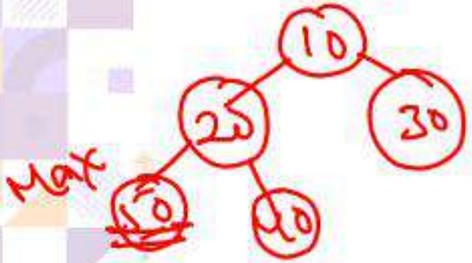
Total No. of nodes = 10

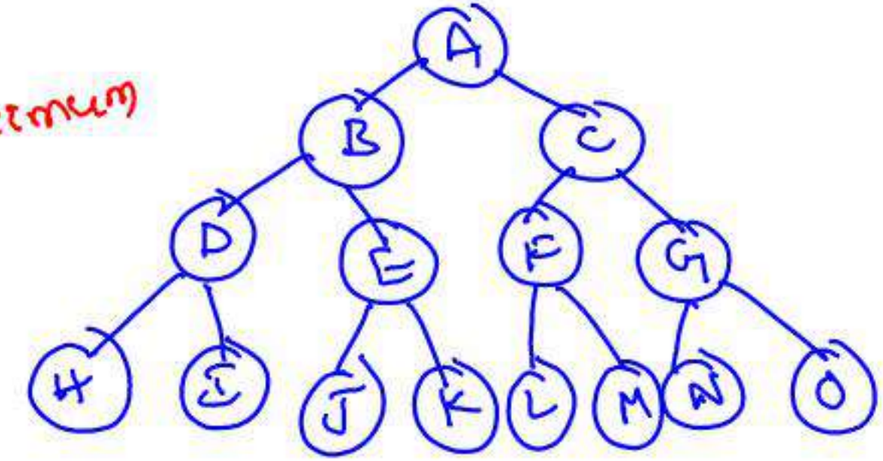
No. of leaf nodes = $\frac{10}{2} = 5$

$\left\lceil \log_2^{10} \right\rceil \Rightarrow 4$ ✗



(OR)





Total No. of nodes = 15 = n

No. of leaf nodes = $\left\lceil \frac{15}{2} \right\rceil = 8$

$\log n = \left\lceil \log_2^{15} \right\rceil = 4$ ✗

BYJU'S EXAM PREP

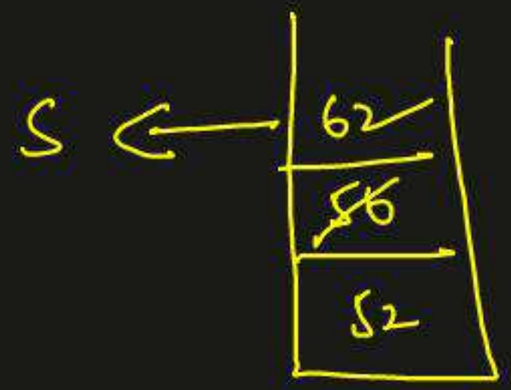
Q3. Consider the sequence of operations given below S + R =_____ .

Push(52)
Push(56)
POP()
Push(62)
S = POP()

ENQUEUE(43)
ENQUEUE(24)
DEQUEUE()
ENQUEUE(57)
R = DEQUEUE()

$S + R = 62 + 24$

$= 86$



BYJU'S EXAM PREP

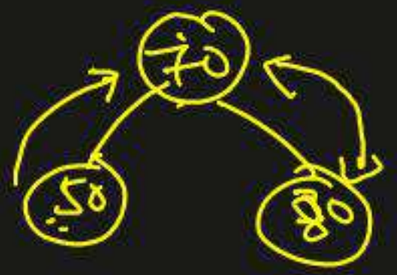
Q4. Consider a Binary search Tree T; Time to find element smaller than maximum element in T ?

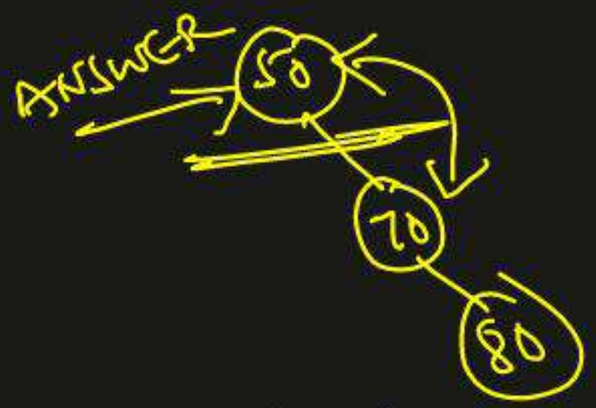
A. O(n)

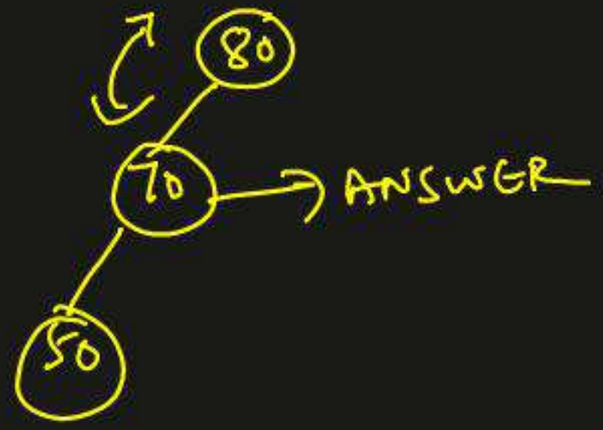
B. O(1)

C. O(nlogn)

D. O(logn)







Constant time Complexity

BYJU'S EXAM PREP

Q5. Evaluate and write the result for the following postfix expression :

abc*+de*f+g*+ where a=1, b=2, c=3, d=4, e=5, f=6, g=2.

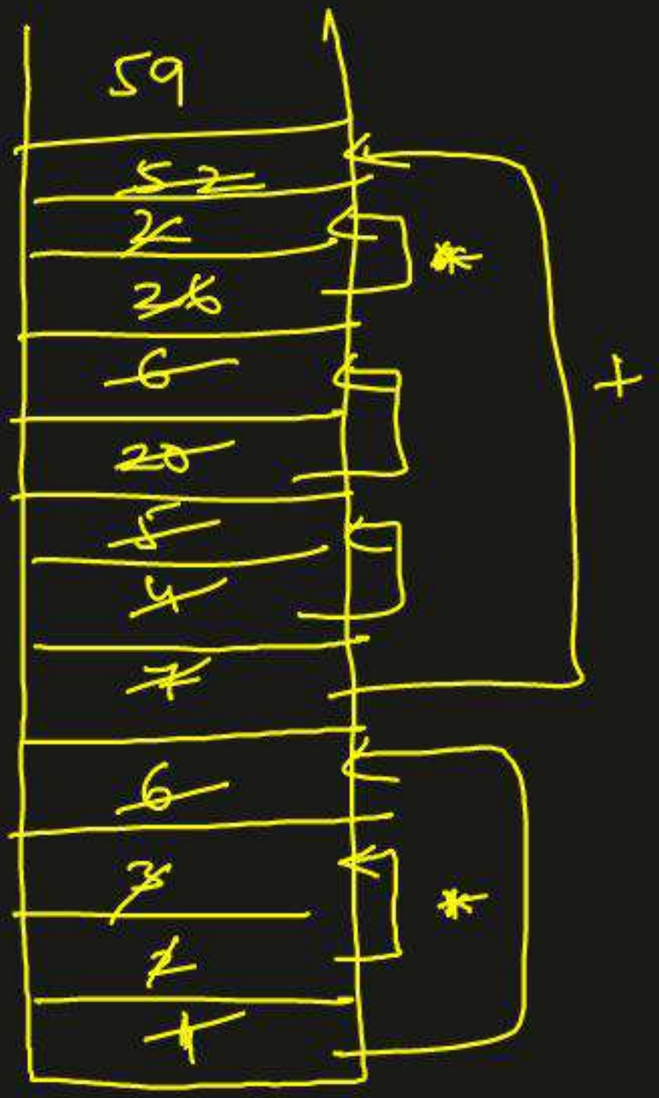
A. 61

B. 59

C. 60

D. 55

1 2 3 * + 4 5 * 6 + 2 * +



BYJU'S EXAM PREP

Q6. Which of the following permutation can be obtained in the same order using a stack assuming that input is the sequence 5, 6, 7, 8, 9 in that order?

A. 7, 8, 9, 5, 6

B. 5, 9, 6, 7, 8

C. 7, 8, 9, 6, 5

D. 9, 8, 7, 5, 6

Wrong i/p

(A) Push 6, Push 5, Push 7, Pop, Push 8, Pop, Push 9, Pop, Pop, Pop 7, 8, 9, 5, 6

(B) Push 5, Pop, Push 9, Pop, Push 6, Pop, Push 7, Pop, Push 8, Pop 5, 9, 6, 7, 8

(C) Push 5, Push 6, Push 7, Pop, Push 8, Pop, Push 9, Pop, Pop, Pop 7, 8, 9, 6, 5

BYJU'S EXAM PREP

PYQ

Q7. Convert the following infix expression to postfix expression

B * C - C + D / A / (E + E)

A. B C * C - D A / E E + / +

B. B C C - * D / A E E + / +

C. B C C - * D A / E E + / +

D. B C C * - D A / E E + / +

Stack

*	*	+	+	/	(	+	+

Postfix exp: B C * C - D A / E E + / +

BYJU'S EXAM PREP

Q8. The prefix form of A-B/ (C * D ^ E) is ?

) = ((=)

A. -/*^ACBDE

B. -ABCD*^DE

C. -A/B*C^DE

D. -A/BC*^DE

Stack :

(	^	*	/	-	

Postfix : E D ^ C * B / A -

Reverse result ⇒ Prefix expression : - A / B * C ^ D E

BYJU'S EXAM PREP

Q9.

```
A(int n)
{   if(n<=1)  return;
    else {
    A( n-2);
    printf(n);
   A(n-1);
   printf(n-2);  }
}
```

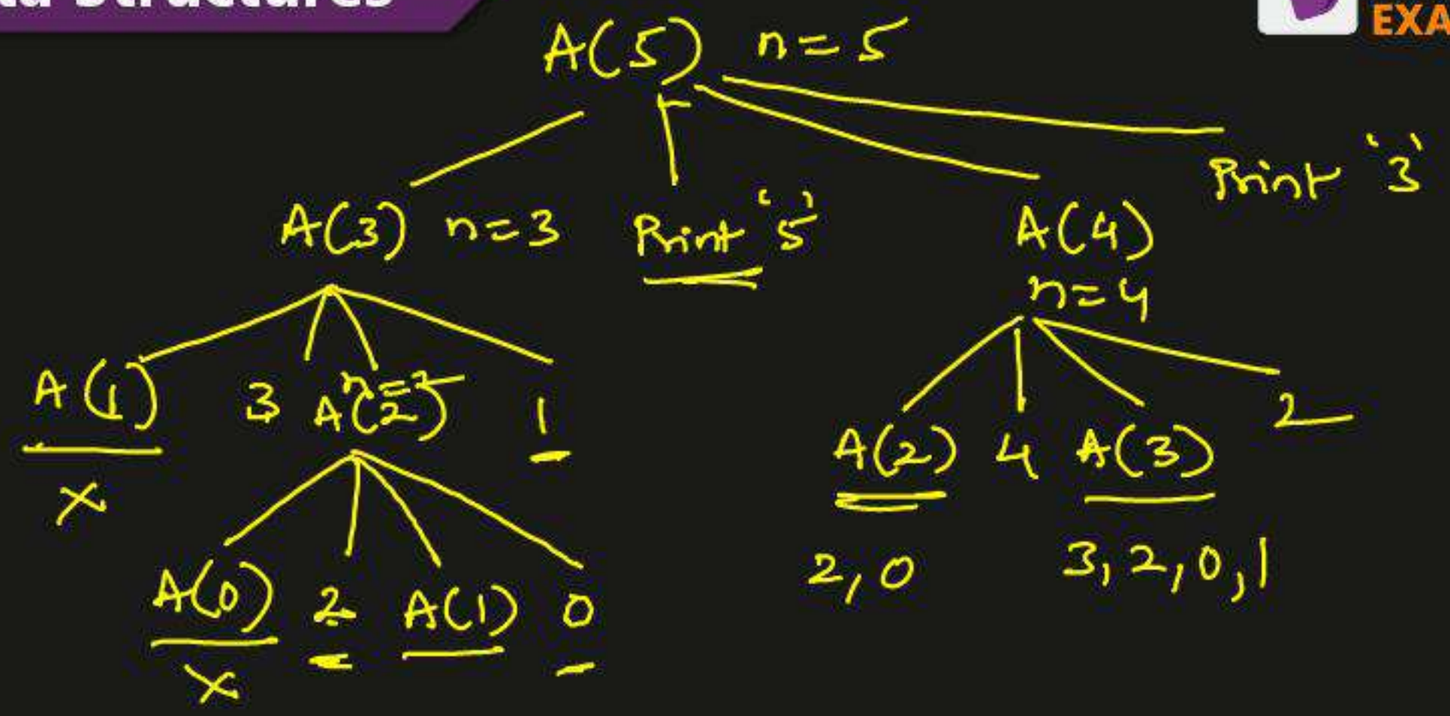
What is output of A(5) ?

A. 3, 2, 0, 1, 5, 2, 0, 4, 3, 2, 0, 1, 2, 3

B. 3, 2, 0, 1, 5, 2, 0, 4, 2, 3, 0, 1, 2, 3

C. 3, 2, 0, 1, 5, 2, 0, 4, 2, 3, 0, 1, 2, 3

D. None of the above



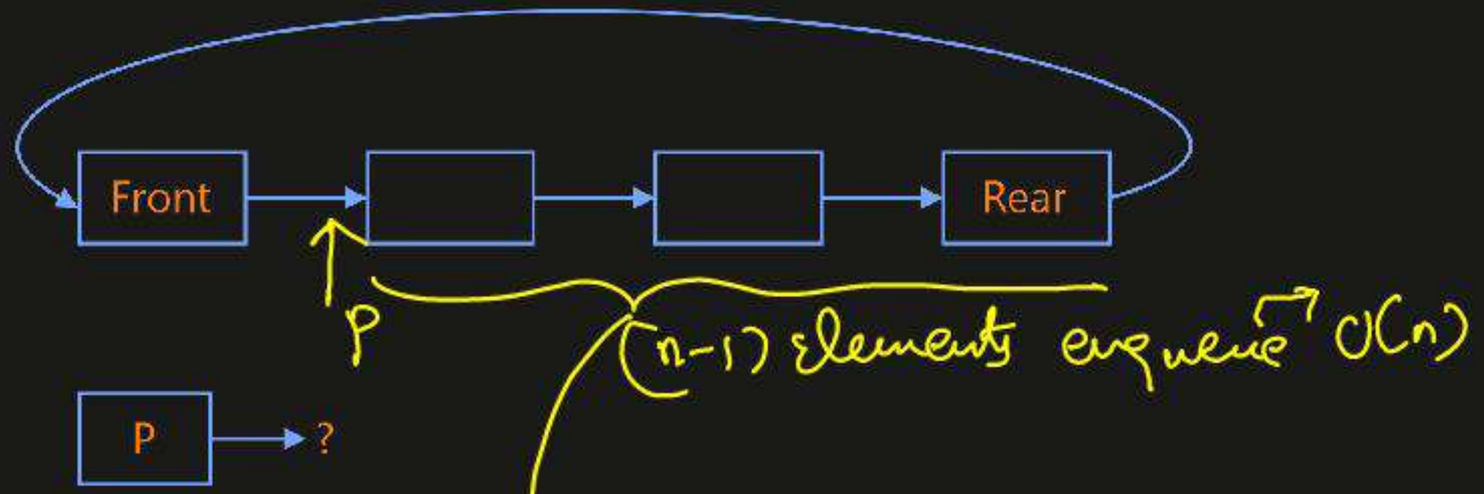
o/p: 3, 2, 0, 1, 5, 2, 0, 4, 3, 2, 0, 1, 2, 3

BYJU'S EXAM PREP

```
A(int n)
{   if(n<=1)  return;
    else {
    A( n-2);
    printf(n);
   A(n-1);
   printf(n-2);  }
}
```

BYJU'S EXAM PREP

Q10. A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?

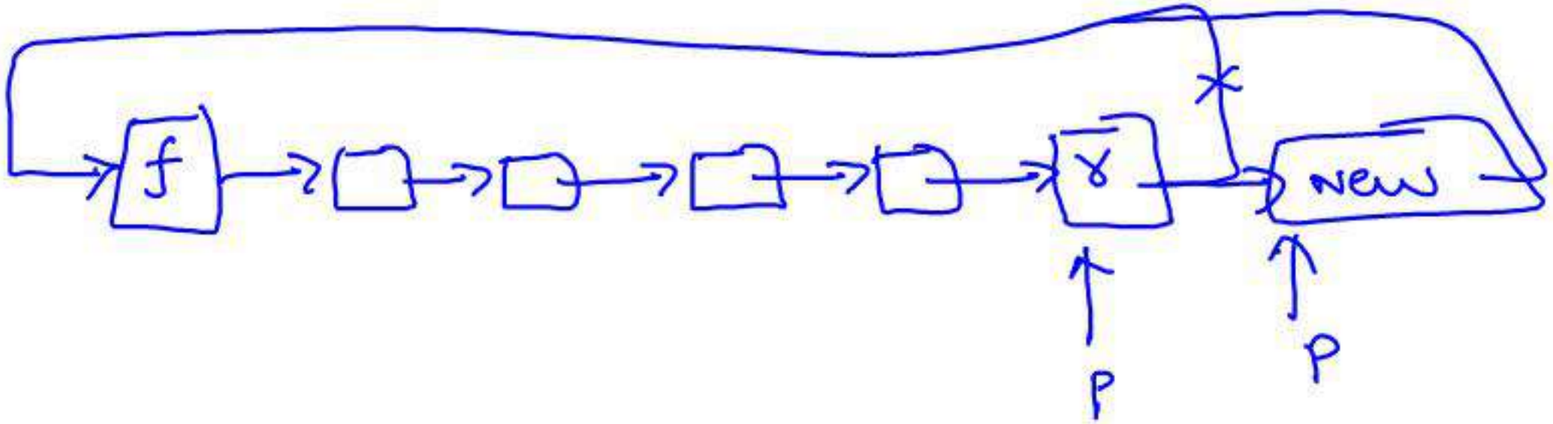


A. rear node

B. front node

C. not possible with a single pointer

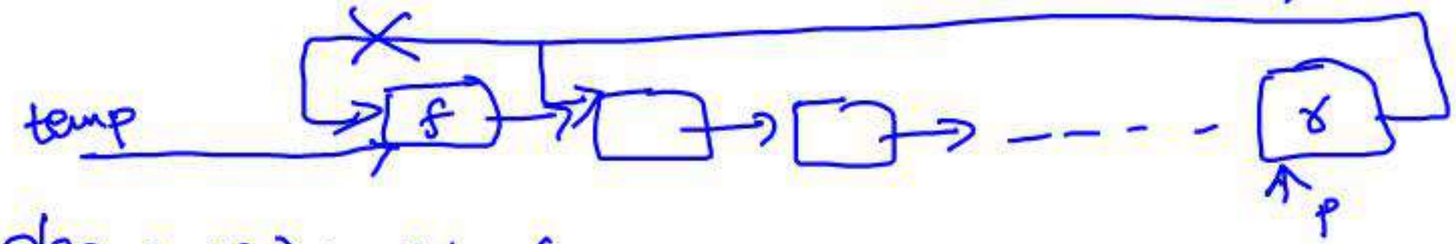
D. node next to front



enqueue( ) : rear End

```
New->next = P->next;
P->next = New;
P = P->next;
```

Constant time operations



dequeue(); At front End

```
temp = P->next;
P->next = P->next->next;  free(temp);
```

Constant time operations

(D) L->next = New;
New->next = NULL;

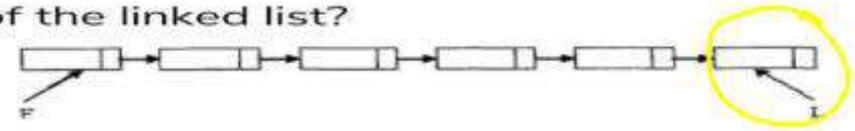
BYJU'S EXAM PREP

1:23 PM

← Quiz: Linked Lists - 1

Q.5 +2, -0.66

Consider a single linked list where F and L are pointers to the first and last elements respectively of the linked list. The time for performing which of the given operations depends on the length of the linked list?



(A) Delete the first element of the list

(B) Interchange the first two elements of the list

(C) Delete the last element of the list
SKIPPED

(D) Add an element at the end of the list

SEE SOLUTION

CORRECT ANSWER : C

Delete last element

free(L) : element gets deleted

Pointer of last but one node still points to the same location. : Dangling Pointer

So, start from F and traverse till second last element and make it NULL

⇒ As No. of elements increases, time also increases. O(n)

BYJU'S EXAM PREP

← Quiz: Queues - 2

SEE SOLUTION

Q.4 +2, -0

Consider a circular array-based queue Q capable of holding 7 elements. Suppose if the following code is executed then what is the value present at the location 3 (if the index starts 0) _____

```
for(k=1; k<=7; k++)
Q.enqueue(k);
for(k= 1; k<=4; k++)
{
        Q.enqueue(Q.dequeue());
        Q.dequeue();
}
```

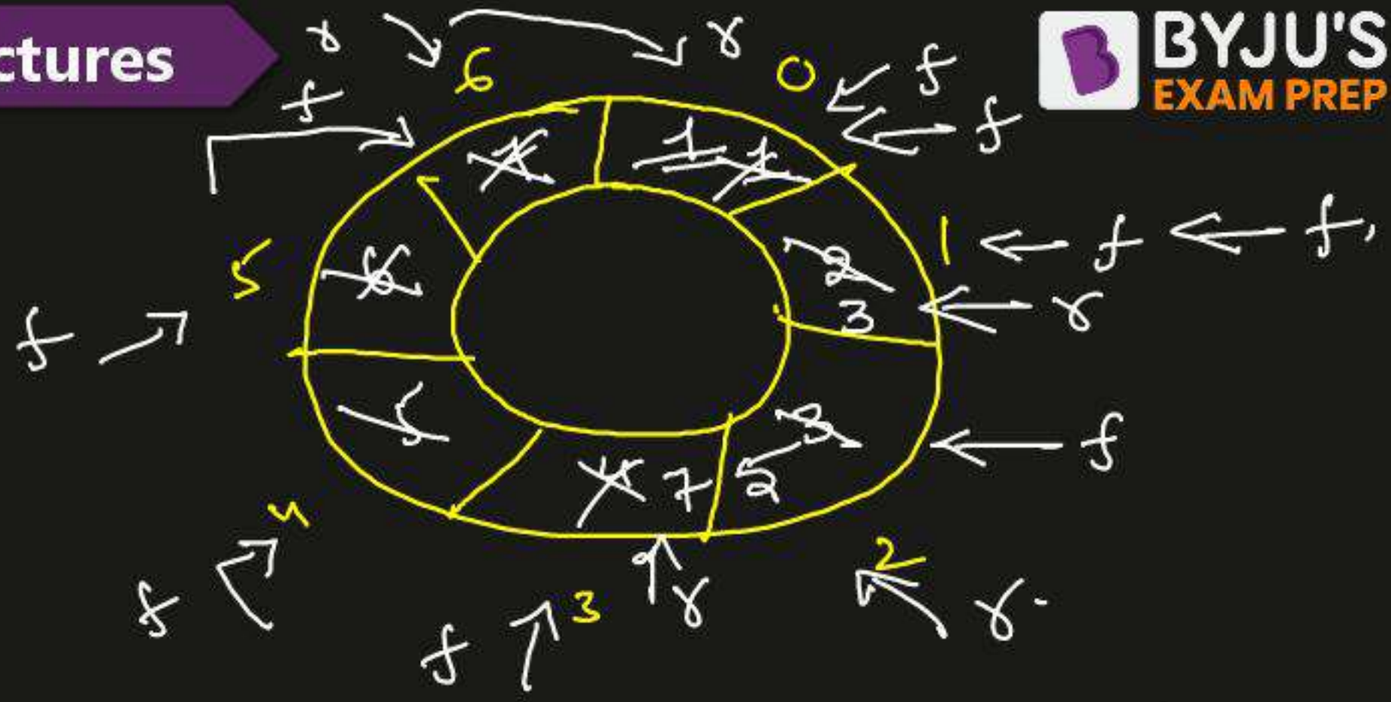
5

SEE SOLUTION

CORRECT ANSWER : 7

After executing the above code we will get final array

Array index	0	1	2	3	4	5	6
k	1	2	3	4	5	6	7
		3	5	7			



k=1 Q.enqueue(1), Q.dequeue()

k=2 Q.enqueue(3) Q.dequeue()

k=3 Q.enqueue(5) Q.dequeue()

k=4 Q.enqueue(7) Q.dequeue()

ANSWER: 7 value at location 3.

BYJU'S EXAM PREP

3:26 PM

Quiz: Linked Lists - 2

What would be the output after the following sequence of steps are executed on the given linked list ?

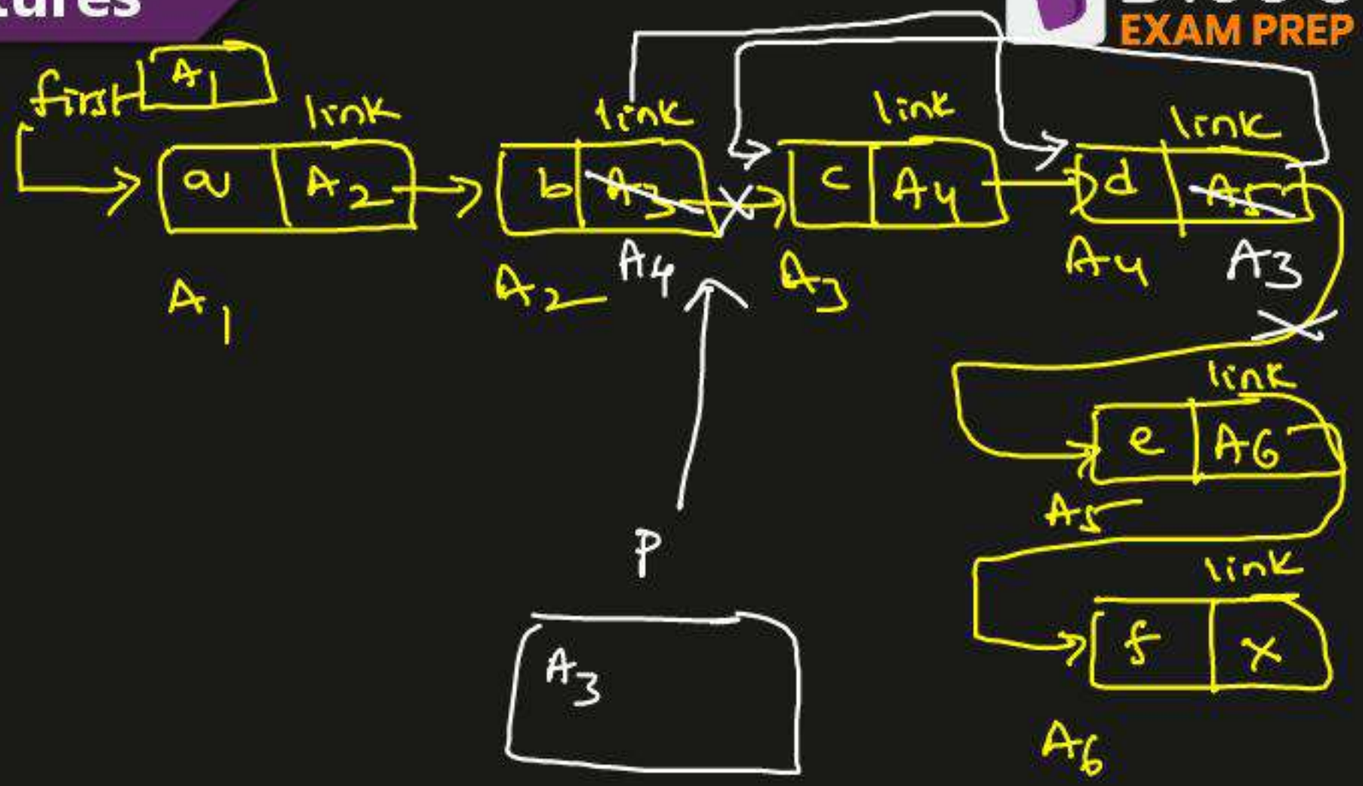
```
struct node *p;

p = first → link → link;

p → link → link = p;

first → link → link = p → link;

printf("%c", first → link → link → link → data );
```

(A) a

(B) b

(C) c

SKIPPED



first → link → link → link → data

A_2 → link ⇒ A_4

A_4 → link ⇒ A_3 → data

C

BYJU'S EXAM PREP

12:47 PM

← Quiz: Pointers and Arrays - 3

Q.6 +2, -0

Consider the following code:

```
void f(int a[ ])
{
int *P[] = {a, a+ 1, a + 2, a+ 3};
int * *q;
q = P;
q + +;
*(+ + q);
+ +(*q);
printf("%d", **q);
}
void main( )
{
    int x[] = {10, 20, 30, 40};
    f(x);
}
```

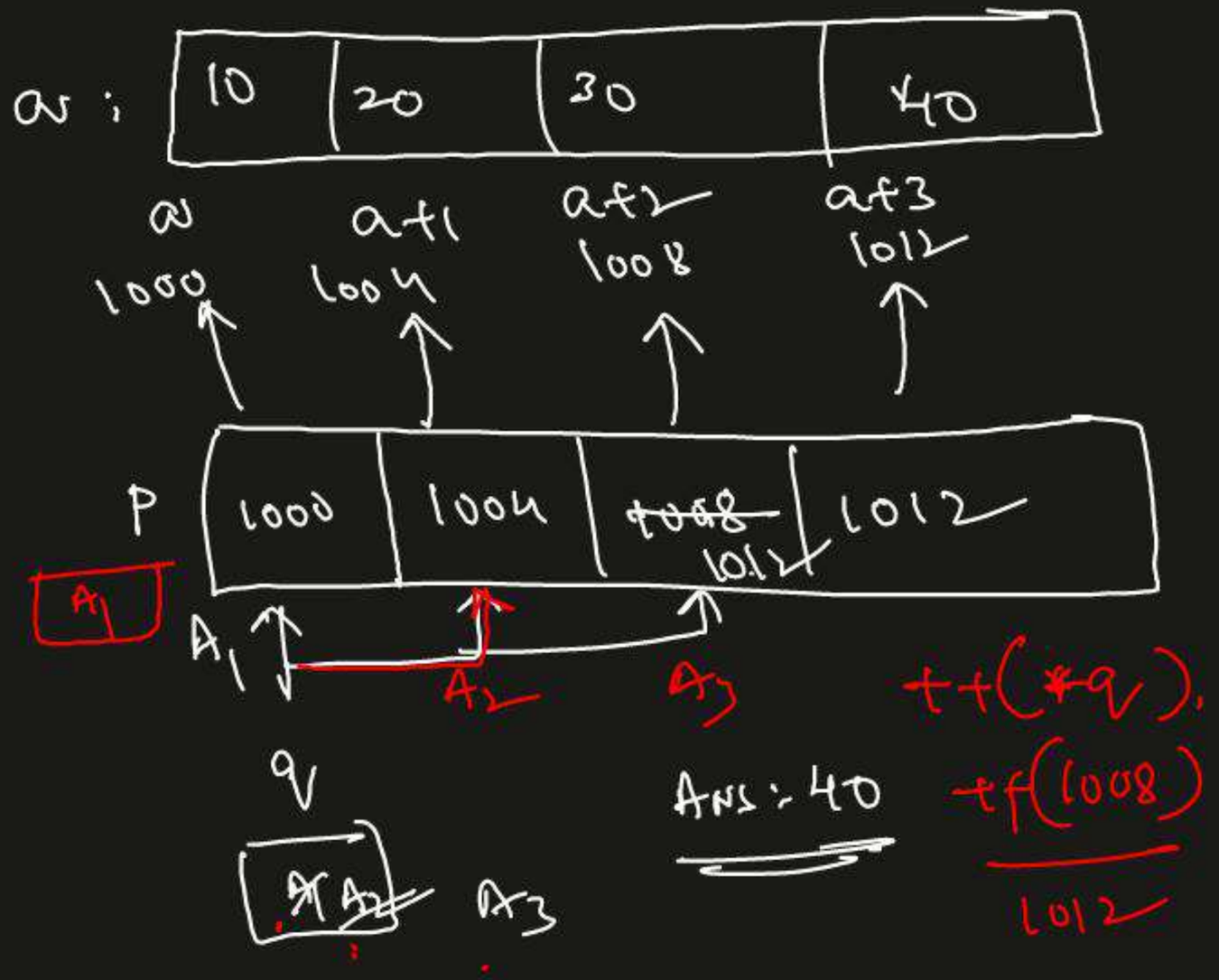
The output of the above program is _____ .

No Response Entered

SEE SOLUTION

CORRECT ANSWER : 40

*(++q)
A3
*(A3) = 1008 → function f.



BYJU'S EXAM PREP

Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry W_{ij} in the matrix W below is the weight of the edge {i, j}.

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

B–A–E–C = 8

B–D–E–C = 9

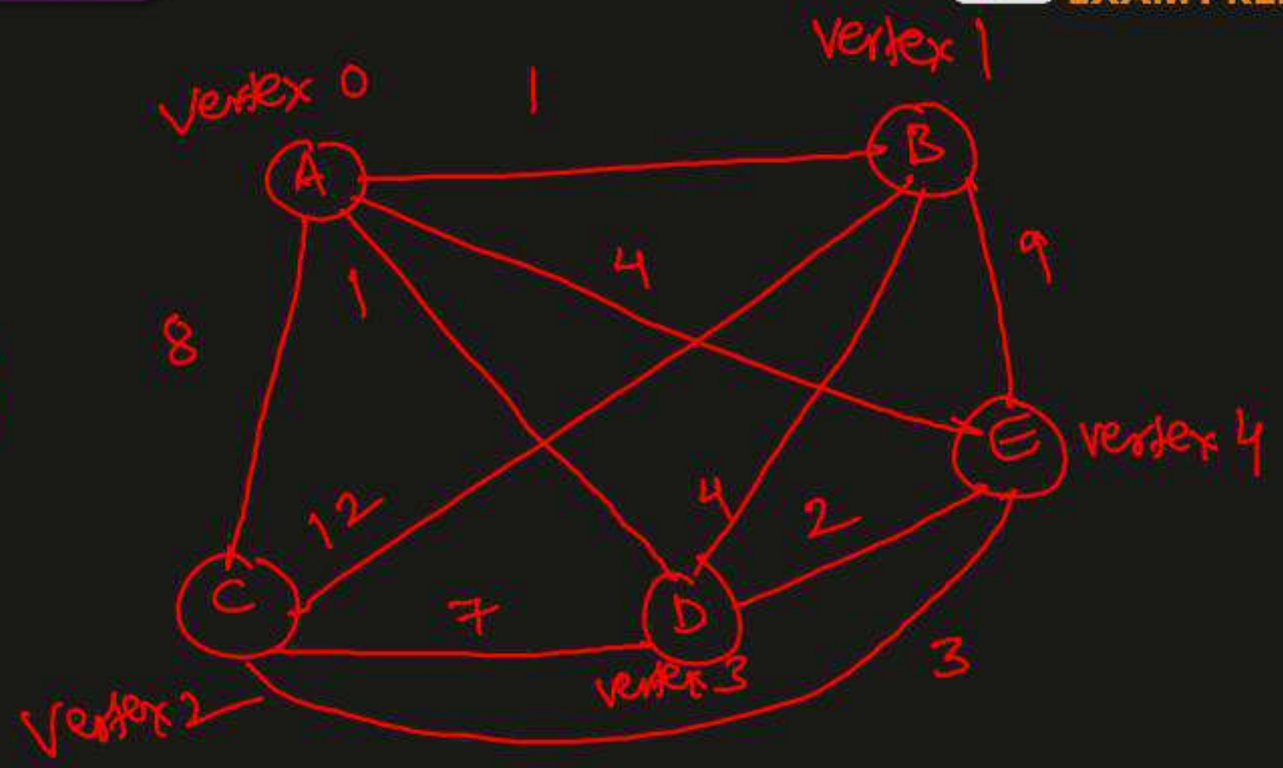
B–E–D–C = 18

What is the minimum possible weight of a path P from vertex 1 to vertex 2 in this graph such that P contains at most 3 edges?

B–D–A–C = 13

A. 7 B. 8

C. 9 D. 10



vertex 1 (B) to vertex 2 (C)

B–A–D–C = 9

B — C = 12

B — A — C = 9

B — D — C = 11

B — E — C = 12

BYJU'S EXAM PREP

Consider the following statements:

S_1: The sequence of procedure calls corresponds to a preorder traversal of the activation tree.

S_2: The sequence of procedure returns corresponds to a postorder traversal of the activation tree.

Which one of the following options is correct?

(a) S_1 is false and S_2 is false.

(b) S_2 is true and S_2 is false.

(c) S_1 is true and S_2 is true.

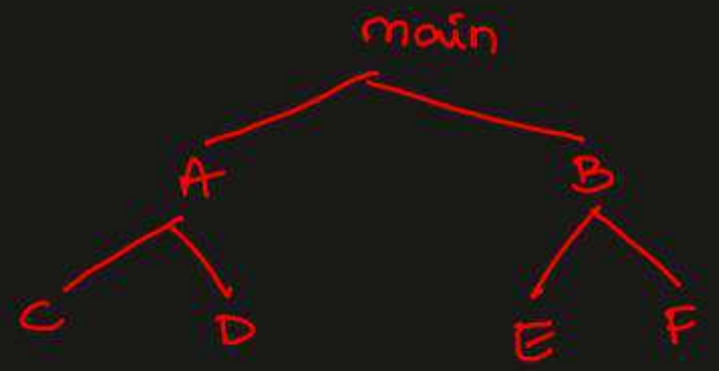
(d) S_1 is false and S_2 is true.

[2021 (Set-1) : 1 Mark]

main( ) { A(); B(); }

A() : { C(); D(); }

B(); { E(); F(); }



Pre order : main, A, C, D, B, E, F

Post order : C, D, A, E, F, B, main.

Sequence of Procedure Calls: main, A, C, D, B, E, F

Sequence of Procedure returns: C, D, A, E, F, B, main

BYJU'S EXAM PREP

What would be returned for the call F (99)? (In integer)

13. Consider the following pseudo code:

```
A(n)                    B(n)
{                       {
if(n<=1) return;        if(n<=1) return;
else                    else
{                       {
    B(n-2);  B(2)           print(n-1);
    print(n);               A(n-1);
    B(n-1);                 A(n-2);
}                           print(n);
}                       }}
```

What will be the product of all the digits printed by this code if A (4) is called?

Practice

BYJU'S EXAM PREP

The number of ways in which the numbers 1, 2, 3, 4, 5, 6, 7 can be inserted in an empty binary search tree, such that the resulting tree has height 6, is _____. ***Note:*** The height of a tree with a single node is 0.

[2016 (Set-2) : 2 Marks]

number of Nodes = 7 = n

height of tree = 6 = n-1

Maximum height

⇒ Left skewed (or) right skewed (or) Equivalent

h=0 (1) (OR) (7)

h=1 (1)–(2) (OR) (7)–(6)

h=2 (1)–(2)–(3) (7)–(6)–(5)

Root ⇒ 2 height 6

⇒ 2 height 5

2 4

2

⇒ 2×2×2×2×2×2

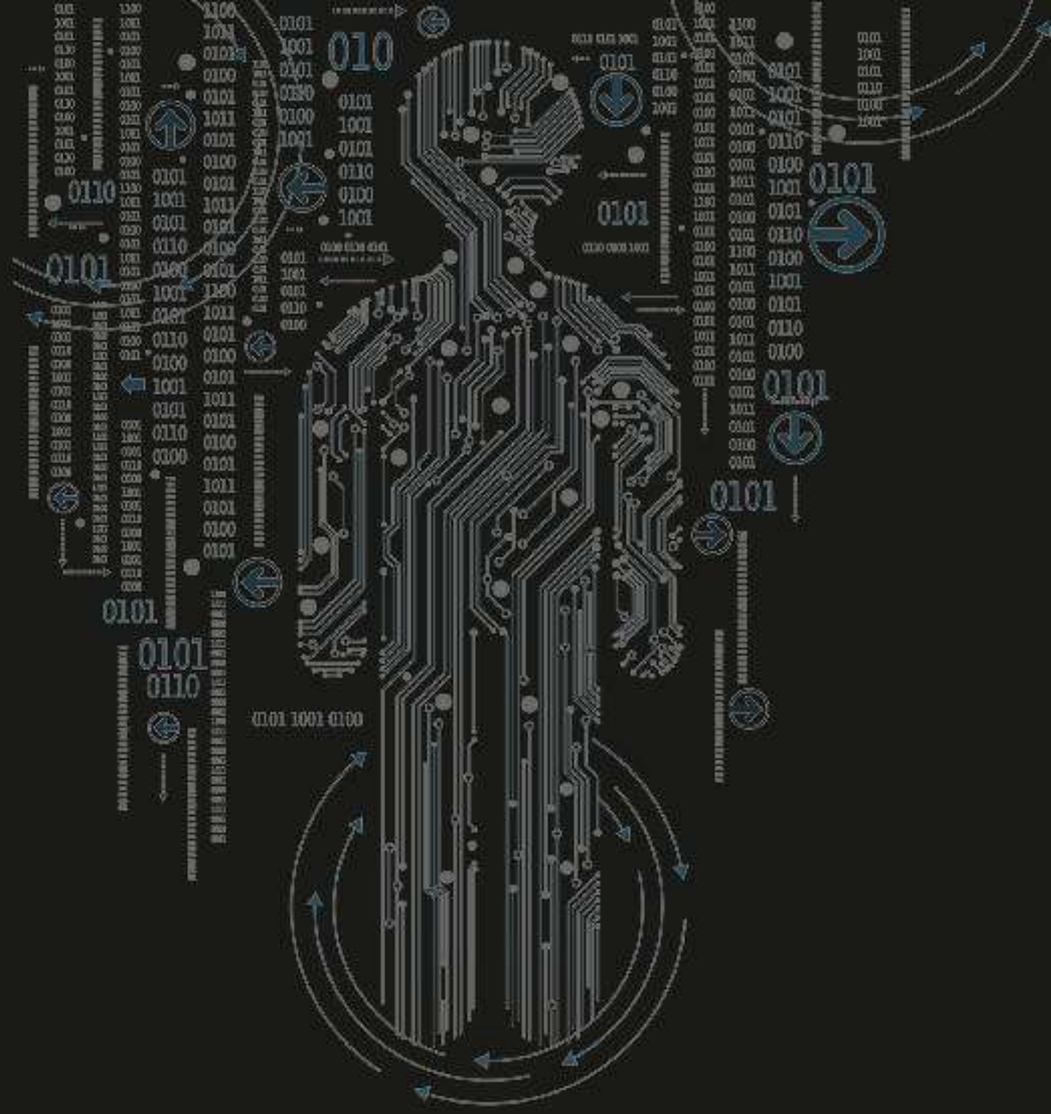
$= 2^6 = \boxed{64}$

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play

BYJU'S EXAM PREP

PROGRAMMING & DATA STRUCTURES

Interactive Session - 2



Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

Interactive Session - 2

Count = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

Q1. Consider the following 'C' code.

The value of count returned for fun(65) is _____ .

```
int fun (int n)
{   int count=0;
        while(n!=1024)
    {
    count++;
    n>>=1;
    n<<=2;
    }
return(count);
}
```

fun(65) n = ~~65~~ ~~32~~ ~~128~~ ~~256~~ 512

65 != 1024 True Count++

n = n >> 1 ⇒ n/2^1 = 65/2 = 32

n = n << 2 ⇒ n * 2^2 = 32 × 4 = 128

128 != 1024 True Count++

n = n/2 = 128/2 = 64

n = n * 4 = 64 * 4 = 256

256 != 1024 True Count++

n = n/2 = 256/2 = 128

n = n * 4 = 128 * 4 = 512

512 != 1024 True, Count++

n = 512/2 = 256

n = 256 × 4 = 1024

1024 != 1024 False

$\boxed{\text{Ans} : 4}$

BYJU'S EXAM PREP

Ans: 23

Q2.
```
int f (int x)
{        int a=1,b;
         if (x==1)
            return 1;
    for(b=1; b<x; b+=2)
    a+=f(b) + f(x-b);
 return a;
}
```

The return value of f(5) = _____

$f(1)$ $x=1$
$=1$

$f(5)$ $x=5$

$a=1$

$b=1$ $1<5$ True $a = a + f(1) + f(4)$
$a = 1+1+13 = 15$

$b=3$ $3<5$ True $a = 15 + f(3) + f(2)$
$= 17 + 5 + 3 = 25$

$b=5$ $5<5$ False

$f(4)$ $x=4$
$a=1$

$b=1$ $a = 1 + f(1) + f(3) = 1+1+f(3) = 1+1+5 = 7$

$b=3$ $a = 7 + f(3) + f(1) = 7+5+1 = 13$

$f(3)$ $x=3$ $a=1$

$b=1$ $a = 1 + f(1) + f(2) = 1+1+f(2) = 1+1+3$
$= 5$

Interactive Session - 2

```
int f (int x)
{        int a=1,b;
          if (x==1)
              return 1;
    for(b=1; b<x; b+=2)
    a+=f(b) + f(x-b);
 return a;
}
```

f(2) x=2

a=1

b=1 a= 1+1+f(1) = 1+1+1=3

b=3 3<2 False

f(2) returns 3

Interactive Session - 2

Q3.
```
int mul(int i)
{
static int count=2;
count=count*i;
return count;
}
void main ( )
{
    int i, j, count=0;
    for(i=1; i<5; i++)
            j=mul(i);
    for(i=j;i>1;i/=2)
  {
            count++;
            printf("%d",count);
  } }
```

What is Output of above program?

mul(1)
Count = 2×1 = 2

mul(2)
count = 2×2 = 4

mul(3)
Count = 4×3

mul(4)
Count = 12×4 = 48

Count = ~~2~~ ~~4~~ ~~12~~ 48

Ans: 12345

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

i=1	i=2	i=3	i=4
j = mul(1)	j = mul(2)	j = mul(3)	j = mul(4)
j = 2	j = 4	j = 12	j = 48

i=48	i=24	i=12	i=6	i=3	i=1
1	2	3	4	5	

Interactive Session - 2

Q4. Z be an array: defined such that

Z[i] =1 for 0<=i<=9

Then the function

```
function(int array[10], int n, int x)
{
int count, var=0;
int total=array[var];
for(var=1; var<=n-1; var++)
{ total = total * x + array[var];
}
return total;
}
```

Called by parameters: function(Z, 10,2)

total = 1023

fun(z, 10, 2) n x

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

Total = 1

var=1 total = 1 * 2 + 1 = 3

var=2 total = 3 * 2 + 1 = 7

var=3 total = 7 * 2 + 1 = 15

var=4 total = 15 * 2 + 1 = 31

var=5 = 63

var=6 = 127

var=7 = 255

var=8 = 511

var=9 = 1023

total

Interactive Session - 2

```
function(int array[10], int n, int x)
{
int count, var=0;
int Total=array[var];
for(var=1; var<=n-1; var++)
{ total = total * x + array[var];
}
return total;
}
```

```
function(Z, 10,2)
```

BYJU'S EXAM PREP

$i = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{3}\ 4$

Q5. Assume following C code,

```
void main()
{
int i, j, count;
count = 0;
i = 0;
for (j = –3; j ≤ 3; j++)
if((j ≥ 0) && (i++))
count = count + j;
}
count = count + i;
printf("%d", i);
return;
}
```

count

Count = 0

$j = -3$	$(-3 \geq 0)$ False && —	False	Count = 0
$j = -2$	$(-2 \geq 0)$ F && —	F	Count = 0
$j = -1$	$(-1 \geq 0)$ F && —	F	Count = 0
$j = 0$	$(0 \geq 0$ True$)$ && (0)	False	Count = 0
$j = 1$	$(1 \geq 0)$ && 1	T	Count = 0 + 1 = 1
$j = 2$	$(2 \geq 0)$ && 2	T	Count = 1 + 2 = 3
$j = 3$	$3 \geq 0$ && 3	T	Count = 3 + 3 = 6

Count = 6 + 4 = 10

Interactive Session - 2

A. Program will compile successfully & after complete execution it will print value. 13

B. Program will not compile successfully.

C. Program will compile successful & prints value 10 after execution.

D. Program will compile successful & print 8.

BYJU'S EXAM PREP

Q6. int main ( )

```
{
int a [4] [5]
int i, j;
for (i = 0) i < 4; i ++)
for (j = 0; j < 5; j ++)
a [i] [j] = 10 * i + j;
printf ("% d", * (a [1] + 9));
```

What is the output printed by above code?

a[1]

	0	1	2	3	4
0	0 100	1 104	2 108	3 112	4 116
1	10 120	11 124	12 128	13 132	14 136
2	20 140	21 144	22 148	23 152	24 156
3	30	31	32	33	34

$*((a+1)+9)$

Ans: 24

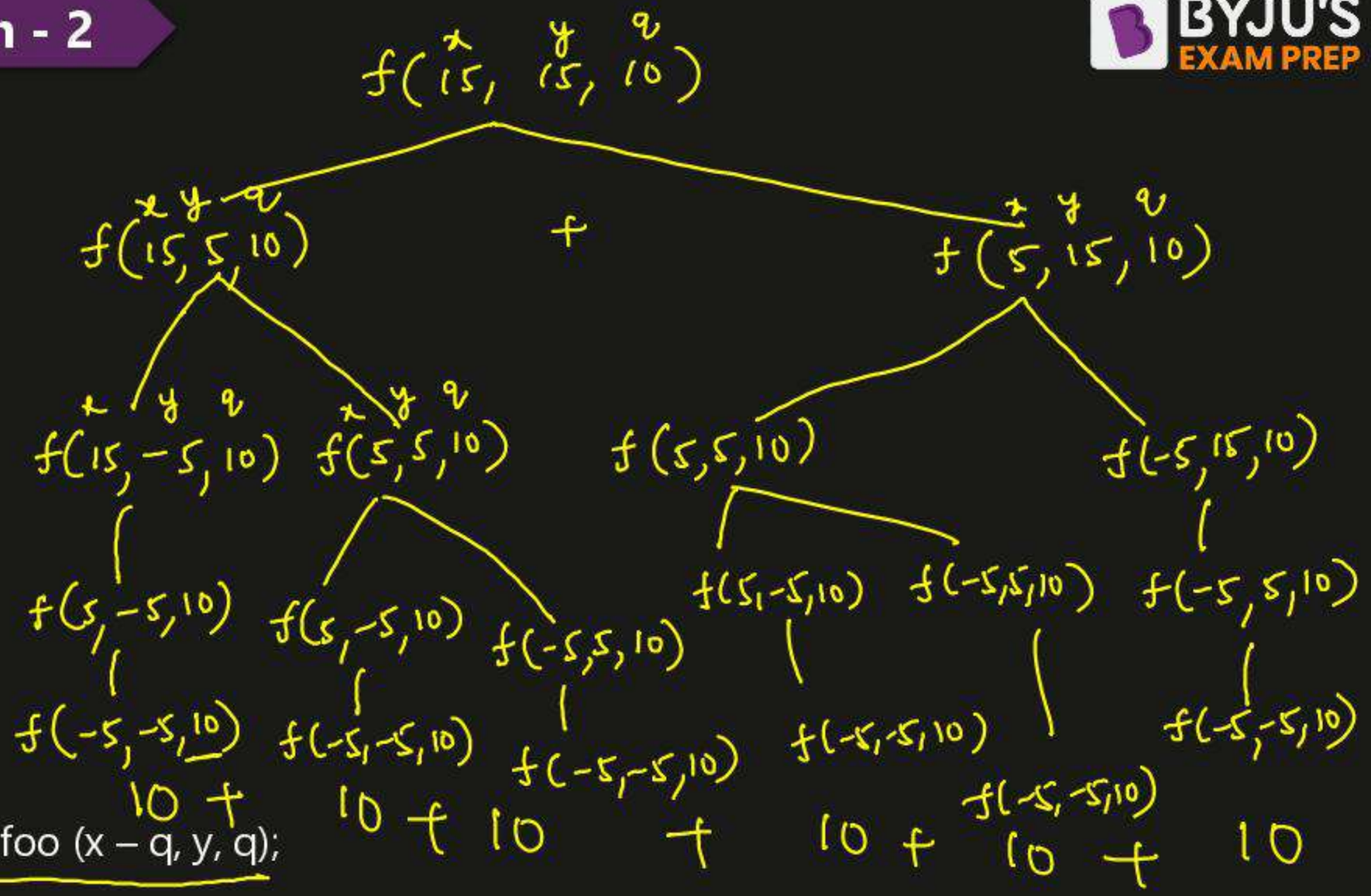
$*(*(a+1)+9)$, $120+9\times4 = *(156) = 24$

BYJU'S EXAM PREP

Q7. int foo(int x, int y, int q)

```
{
if ((x < = 0) && (y <=0))
return q;
else if (x <=0)
return foo (x, y – q, q);
else if (y <= 0)
return foo (x – q, y, q);
else
return foo (x, y – q, q) + foo (x – q, y, q);
}
```

What is the return value for foo(15, 15, 10) ?



= 60

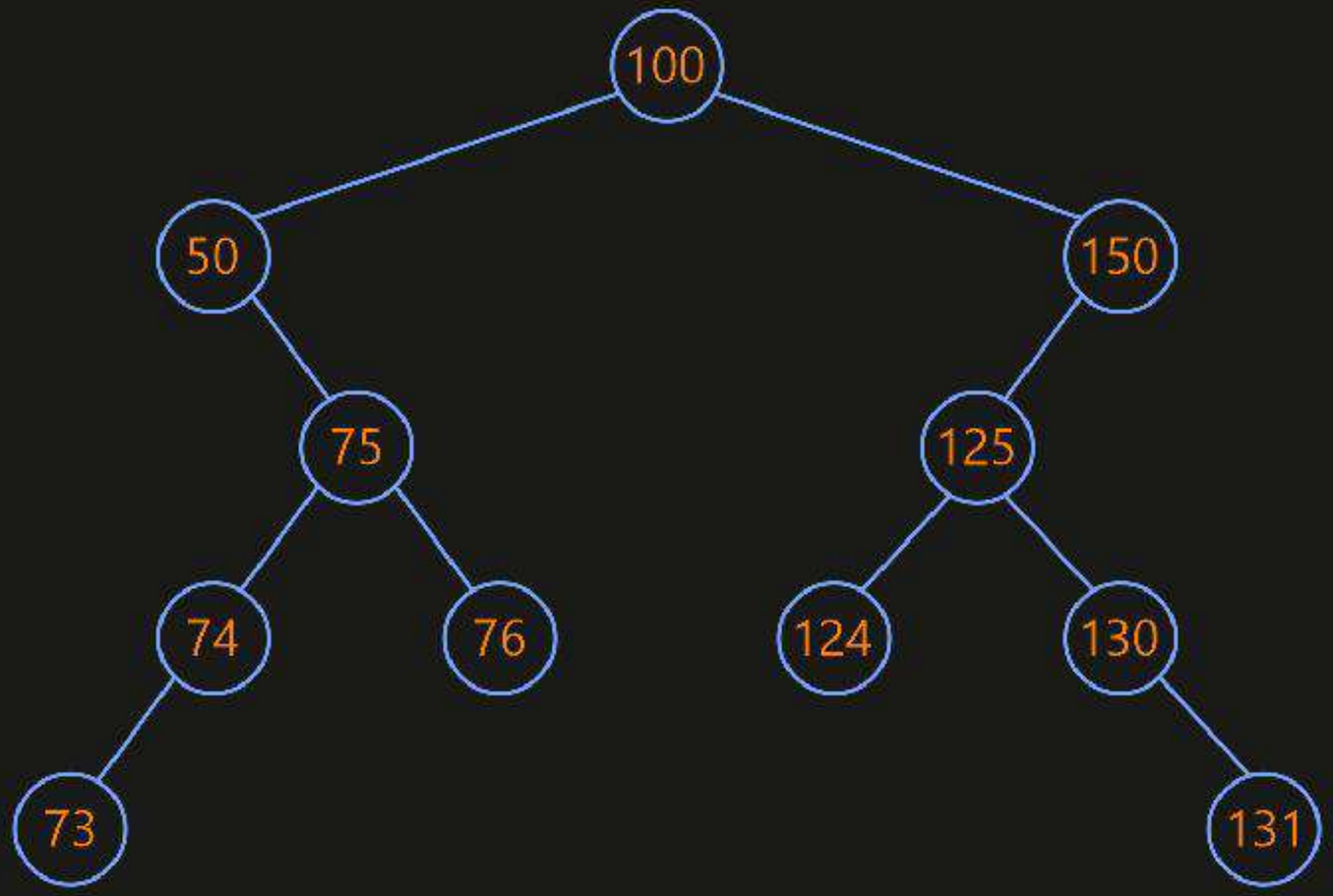
BYJU'S EXAM PREP

Practice

Q8.
```
int foo (int x, int y)
{
if ((x == 1) || (y == 1))
return 1;
if (x ==y)
return x;
if (x > y)
return foo(x – y, y) ;
if (y > x)
return foo(x, y–x) ;
}
```

What is the return value of foo(15, 255) ?

Q9. Pick Incorrect Statement with respect to the given Tree -



Interactive Session - 2

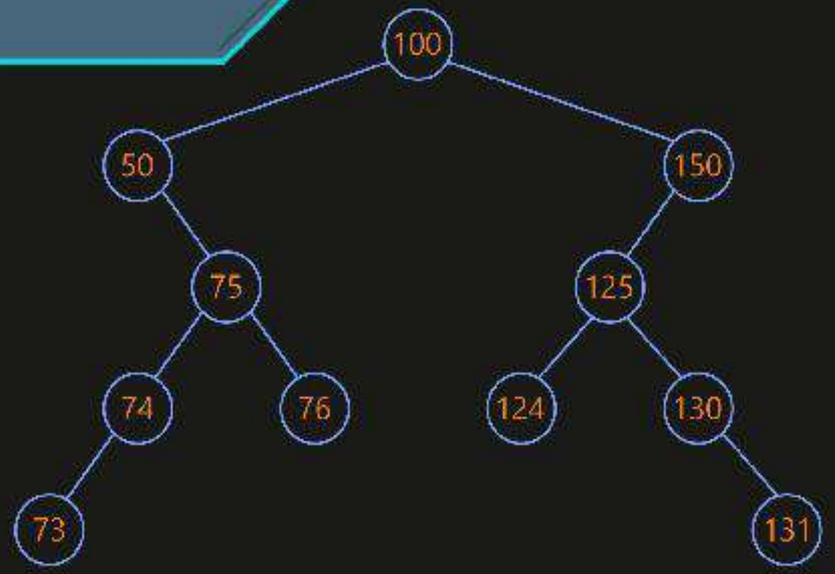
A. Pre Order Traversal : 100, 50, 75, 74, 73, 76, 150, 125, 124, 130, 131.

B. Post Order Traversal : 73, 74, 76, 75, 50, 124, 131, 130, 125, 150, 100.

C. In Order Traversal : 50, 73, 74, 76, 75, 100, 124, 125, 130, 131, 150.

D. The Given Tree is BST.

Practice



BYJU'S EXAM PREP

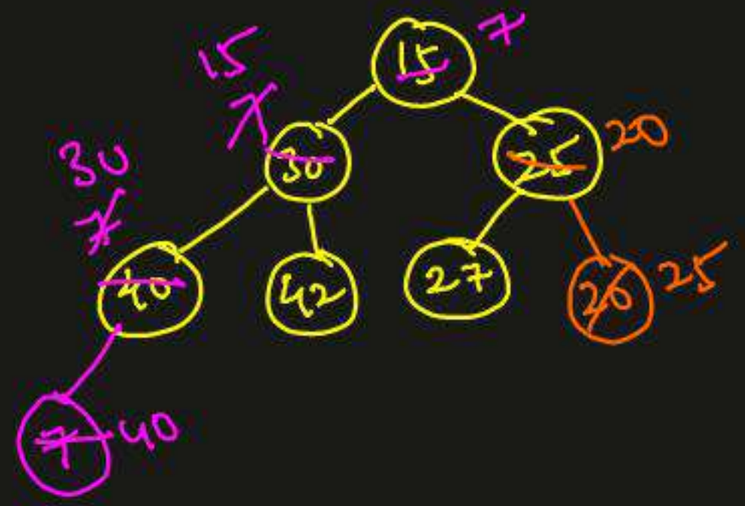
Q10. Consider a min heap with elements 15, 30, 25, 40, 42, 27. After inserting elements 20 and 7, The Level order traversal sequence would be _____ .

A. 7, 15, 20, 30, 40, 42, 27, 25

B. 7, 15, 20, 30, 42, 40, 27, 25

C. 7, 15, 20, 30, 27, 42, 40, 25

D. 7, 15, 20, 30, 42, 27, 25, 40



7, 15, 20, 30, 42, 27, 25, 40

BYJU'S EXAM PREP

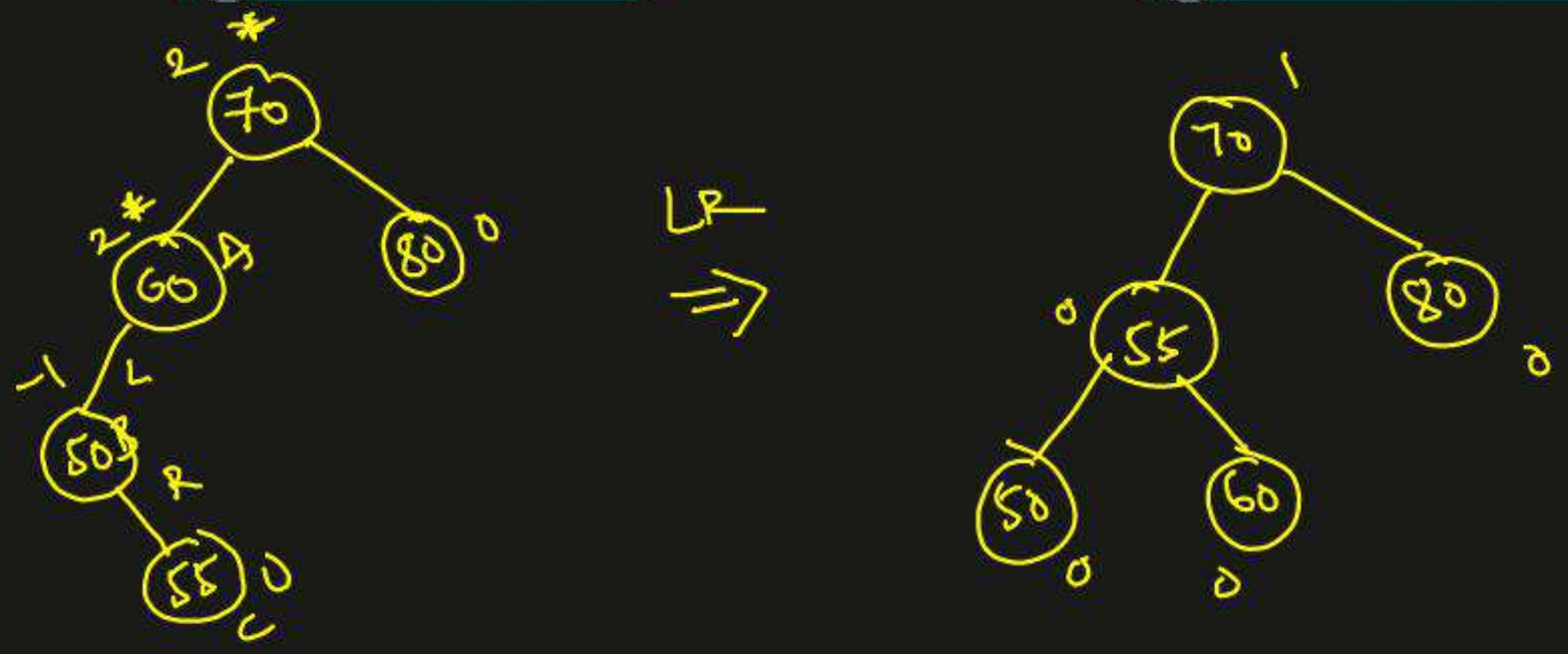
Q11. The Rotation performed to balance BST with elements 70,60,80, 50, 55, would be _____ .

A LL

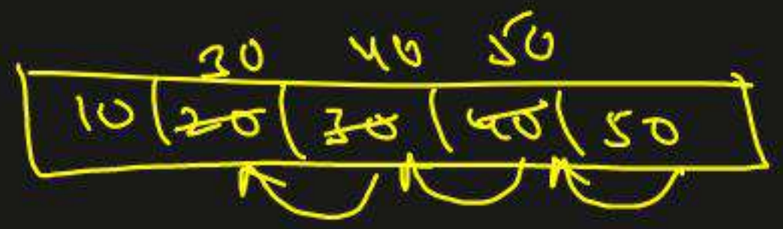
B LR

C RL

D RR



Interactive Session - 2



Q12. Which of the following points is/are not true about Linked List data structure when it is compared with array ?

ANS: D

T **A** Arrays have better cache locality that can make them better in terms of performance.

T **B** It is easy to insert and delete elements in Linked List.

True **C** Random access is not allowed in a typical implementation of Linked Lists.

D Access of elements in linked list takes less time than compared to arrays.

False

Sequential access

Random Access

Interactive Session - 2

We have a binary heap on n elements and wish to insert n more elements (not necessarily one after another) into this heap. The total time required for this is

(A) θ(logn)
(B) θ(n)
(C) θ(nlogn)
(D) θ(n^2)

(A) A
(B) B
(C) C
(D) D

General Insertion

① Add an element
② Heapify

Repeat these steps ① & ② for Every element

→ Insert all at a time

⇒ Add all elements → O(1)
Heapify together for old n + new n
⇒ 2n
= O(n)

Interactive Session - 2

1. Let M be a two-dimensional array declared as follows:

M: array [5 .... 15] [5 .... 20] of integer.

Assuming that each integer takes one memory location, the array is stored in row-major order and the first element of the array is stored at location 200, what is the address of the element A[i][j]?

A. 11i + j + 140

B. 16j+i+115

C. 16i+j+115

D. 11j + i + 140

$B + [(i-5) * c + (j-5)] * n$

BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play