

C Programming

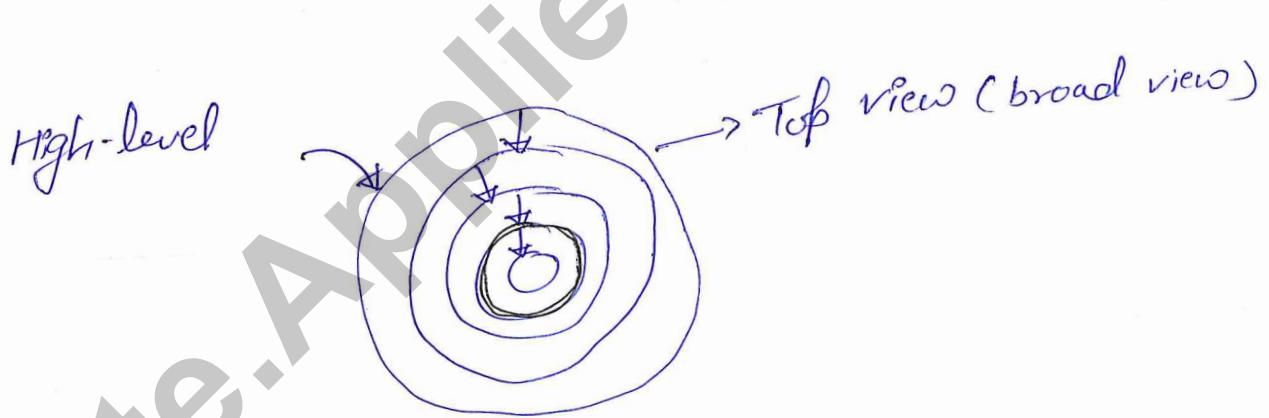
Phone: +91 844-844-0102

Introduction to C Programming

1.1 C: what, why and how

(1) Basic Assumptions: don't know programming
 Some programming
 get better
 Basic

(2) Peeling an onion: Layer-by-layer



Start with each topic and dive completely into it.

(3) Coding: online tool → Linux, Mac, Windows

Share code → Googledoc → email → execute

What is programming?

It's the sequence of step by step or precise instructions that give to a computer to perform a specific task.

It is very similar to English

Like English is a way to communicate with people & people have intelligence. Similarly languages are to communicate between people & computers.

Why C?

For understanding C++, you need to learn C. Every programming language has its role.

javascript (JS) is very popular for building websites or run code in browsers.

For ML & AI course, Python is popular.

For writing android app, Java is used

all the operating systems like android, windows,

Linux, mac, ios are built using C/C++

Mail: gatcse@appleroots.com

Kernal : heard of operating system. Phone: +91 844-844-0102



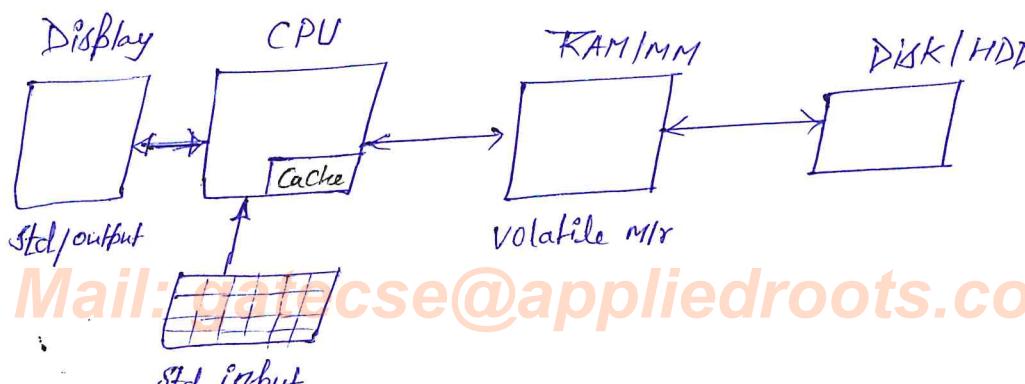
Lot of scientific computing, C/C++ is used.

AI/ML [Tensorflow] → built using C/C++
Weather forecasting

In general, C/C++ code is very faster than a code written in Java, Python, JS and PHP.

- ⇒ And also gaming softwares are also written using C/C++. (has to be faster)
- ⇒ Drivers are also written in C/C++.
- ⇒ hardware/IOT is also designed using C/C++.
- ⇒ If you want to learn foundation of computer science and want a very fast our code.
- ⇒ If speed of execution matters, then C/C++ is probably the best choice.

[1.2] Simplified model of a computer



bask.com (for image)

- ① Chip  → CPU (Central Processing Unit)
↓
It does all the computation
- ② RAM slot → (Thin horizontal strip)
(Main Memory)
- ③ Disk (Hard Drive) : used to store data
- ④ FAN for cooling or for dissipating heat
- ⑤ PSU → connected to electrical socket
- ⑥ CD-ROM →
- ⑦ MotherBoard → Everything is connected to this.
- ⑧ Video Cards → are useful to run game faster.
↳ GPU

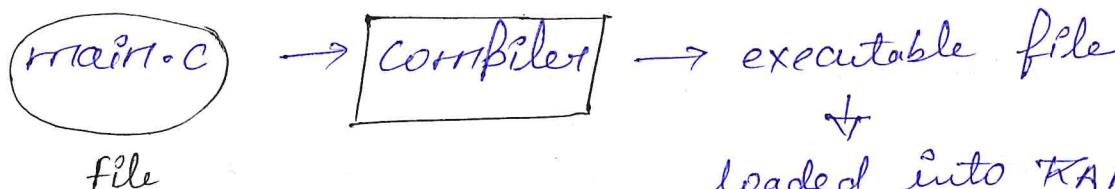
1.3 Tools for Coding

Phone: +91 844-844-0102

5



⇒ extensions for c programs ⇒ .c



GCC → GNU C compiler
 → most popular C compiler

Display
Input

GCC comes prebuilt in Linux, Mac.

⇒ use online compiler, easier/faster to

① onlinegdb.com ② ideone.com

Basics with examples

[2.1] Input and Output

→ full chapter

Objective: high level overview of key concept

Not rigorously

std::input → keyboard	}
std::output → Monitor	

functional

main() {

printf("Hello World");

→ Does not follow many good practices

→ Shows on screen

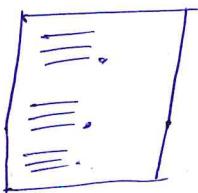
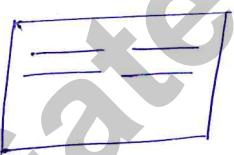
Mail: gatese@appliedroots.com

gots the main func

O/P: hello world

Phone: +91 844-844-0102

- ⇒ all execution of c programs starts from main.
- ⇒ printf means print formatted.
- ⇒ Whatever I want to print in printf, just put it in double quotes (" ") .
- ⇒ If you miss ; after printf.
error: expected ; before { . }
- ⇒ If you forget) in printf statement
error: expected ')' before { token.
error: expected ';' before { token. }
- ⇒ Syntax → Rules of C
- ⇒ Print formatted means printing in some format



```
main ()  
{  
    printf ("hello World It I am here");  
}
```

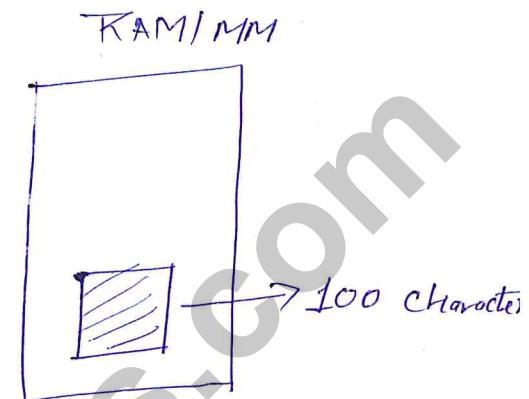
O/P: hello world I am here
Mail: gatecse@appliedroots.com

\n → New line

Phone: +91 844-844-0102

APPLIED
ROOTS

```
main()
{
    char n[100];
    printf("What's your name");
    scanf("%s", n);
    printf("Hello, %s", n);
}
```



O/P: What's your name reveria
hello, reveria

Monitor std o/p
KB (std i/p)

[2.2] choosing one option versus another → C/C++/java

e.g. Imagine MCQ
Question
a.
b.
c.
d.

choice
} If correct ans = a
and your ans = a
then marks
else dont mark

e.g.
hi hello
[B1] [B2]

If B1 is pressed print hi
and B2 is pressed print hello

Mail: gatecse@appliedroots.com

**APPLIED
ROOTS**

```
main() {
    int a;
    printf("enter a number");
    scanf("%d", &a);
    if (a%2 == 0) {
        printf("even");
    } else {
        printf("odd");
    }
}
```

($\&a$)

address of a

O/P: enter a number 12 $12 \div 2$ remainder = 0
 so even

```
main() {
    int a;
    printf("enter a number:");
    scanf("%d", &a);
    if (a%3 == 0) {
        printf("remainder is 0");
    } else if (a%3 == 1) {
        printf("remainder is 1");
    } else {
        printf("remainder is 2");
    }
}
```

O/P: enter a number: 7
 remainder is 1

Mail: gatcse@appliedroots.com

[Q.3] Repeating over and over

Phone: +91 9844041012
Machines can repeat over & over unless humans, they tired

Task: if a is prime no. or not.

Passwords, banking \rightarrow encrypted

(e)

\Rightarrow Computers are brilliant to repeat over & over.

\Rightarrow Prime no. are used for encryption.

\Rightarrow Cryptography: Prime no. are core of cryptography.

$$a = 12$$

logic: $2, 3, 4, \dots, 11$

\uparrow
2

\uparrow
 $a-1$

If 12 is divisible by one number then it's not prime.

exactly divide $a \% i = 0$

\uparrow
 $2, 3, 4, \dots$

repeatedly checking for $2 \dots 11$

$$12 \% 2 = 0$$

12 is not prime

Ex:

$$a = 13$$

$i = 2, 3, 4, \dots, 12$

$$13 \% 2 = 1$$

$$13 \% 3 = 1$$

$$13 \% 4 = 1$$

$$13 \% 5 = 3$$

⋮

$$13 \% i \neq 0$$

$\therefore 13$ is prime no.

repeating \rightarrow Loops

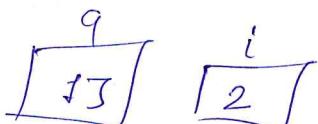
for choices \rightarrow If-else
for repetition \rightarrow for loops



```
main()
{
    int a;
    int i;
    Printf(" enter a number:");
    Scanf("%d", &a);
    for (i=2; i<=a-1; i++) {
        if (a % i == 0) {
            Printf(" NOT PRIME, divisible by %d", i);
            exit(0);
        }
    }
    Printf(" PRIME");
}
```

output

```
for (i=2; i<a-1; i++)
    if 3 % 2 = 1
```



for good programming : logic \rightarrow Write class the logics in english

(74)

01844-844-0102



Convert it C (Syntax)

[2.4] Reuse & Simplify

Task : input two num \Rightarrow input est num \rightarrow Prime or not
input and num \rightarrow prime or not

Logic : input est num \rightarrow 3 lines

{ for loop to test if num is prime or not \rightarrow 6 lines
input and num \rightarrow 3 lines

{ for loop to test if num is prime or not \rightarrow 6 lines

isPrime(a)
I : Prime
O : not Prime

a-P = isPrime(a);

if ($a-P == 1$) {

printf("x.d is prime", a);
}

else {

printf("x.d is NOT a prime", a);
}

Mail: gatecse@appliedroots.com

main()
{
int a;
int a-P;
printf("enter a number");
scanf("%d", &a);
a-P = isPrime(a);
if ($a-P == 1$)
printf("x.d is prime", a);
else
printf("x.d is not prime", a);
printf("1/7 enter another num.");
scanf("%d", &a);
}

O/P: Enter another number: 13
APPLIED ROOTS Phone: +91 844-844-0102

13
15

#include <stdio.h>

```
int isPrime(int a){  
    int i;  
    for(i=2; i<a-1; i++){  
        if(a%i == 0){  
            return 0;  
        }  
    }  
    return 1;  
}  
main()  
{  
    int a;  
    int a_P;  
    printf("enter a number:");  
    scanf("%d", &a);  
    a_P = isPrime(a);  
    if(a_P == 1){  
        printf("%d is Prime", a);  
    }  
    else{  
        printf("%d is NOT a prime", a);  
    }  
}
```

if a is prime
NOT
isPrime(15)

Core basics :
Input and output
choice of if-else/
repeat (for)
reuse (function)
file - disk → pointers

Mail: gatcse@appliedroots.com

[3]

Lets dive in

Phone: +91 844-844-0102

We will start from basics & dive into it to the broad overview.

[3.1] character set in C

Set of characters, that are used in C.

A, B, C - - - Z } alphabets
 a, b, c, - - - z }

0, 1, 2 - - - 9 } digits

Space, newline } white spaces

for ex:

→ If after printing, go to the new line abcd you want to that is white space.
 → New line is a character, which take you to the new line.

⇒ a b : a space b

a b : a tab b (multiple spaces)

+, *, <, >, \$, ! → Special char

⇒ just google as special characters in C.

Mail: gatcse@appliedroots.com

14)

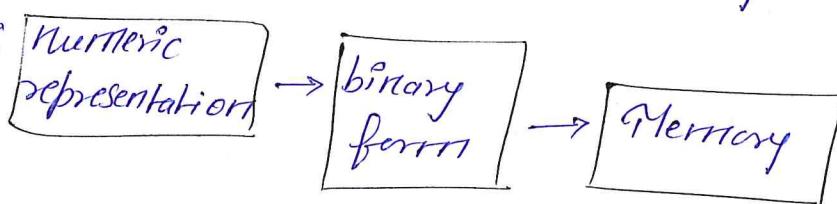
ASCII - Oldies → 0 to ~~255~~ ~~256 numbers~~ 01844-844-0102

Unicode - Modern → 1, 114, 112

Large character set

A → Number → binary → Memory - RAM/disk

Character :



- ⇒ C uses ASCII codes. There are some modern compilers which use unicode.
- ⇒ just google ascii characters in C.

$$A \rightarrow 65$$

$$0 \rightarrow 48$$

$$a \rightarrow 97$$

} Standard mapping

⇒ TAB is also printable character

⇒ characters from 128 to 255 is not printable char

[3.2] Keywords & Identifiers

⇒ Words are basically combination of several characters.

Words → Keywords → 32 predefined words in C
 Which we can not use for any other purpose.
 Ex: if, else, int, char, while

⇒ Google search for keywords

Main: gatecs@appliedroots.com

45

Identifiers: Variables, function **Phone: +91 844-844-0102**

- ⇒ These are words that we use for variable name, funcⁿ name
⇒ Identifiers are valid words in c.

- (1) Alphabets, - (underscore), digits are allowed.
- (2) First character in identifier needs to be an alphabet (U/L). And an underscore. It can not be a digit.
- (3) You can not use keywords as identifiers or variable name.
- (4) These are case-sensitive.

$$abc = 10$$

$$aBC = 20$$

- (5) length of Identifier is compiler/implementation dependent.

→ odder (8)

→ ANSI C

→ 31 char

gcc - visual studio (tool of Microsoft)

Ex: Invalid Identifiers:

- (1) abc X
47 space
- (2) ab!cd X
- (3) 2ab X
- (4) if X

3.3] Data Types

Phone: +91 844-844-0102

⇒ Fundamentally there are four types of data types

- (1) char to store characters.
- (2) int to store integers.
- (3) float → single precision
- (4) Double } store real values.
Double precision

void Data type

- ⇒ It means nothing
- ⇒ useful when we implement functions

ex:

char a; It means in RAM you are creating some space which you are accessing using a. And you can store one char in it.

⇒ There are two types of qualifiers

- (1) short long
- (2) signed, unsigned

⇒ Secondary types of data types

array, string, pointers, structures, enumerators

⇒ No. of bytes used to represent a character depends on the actual system and compiler.

Byte → 8-bits → $\overbrace{\quad\quad\quad}, \overbrace{\quad\quad\quad}$
 $\overset{0/1}{\text{bit}}$ bit

in each space, you can store either 0 or 1.

Data types	Range	Byte	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 ...47	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to 3.4e38	4	
double	-1.7e308 to 1.7e308	8	.f
long double	-1.7e4932 to 1.7e4932	10	.lf
			.lf

=

$$\begin{array}{c} \overline{0} \quad \overline{0} \\ \overline{0} \quad \overline{1} \\ \overline{1} \quad \overline{0} \\ \overline{1} \quad \overline{1} \end{array} \left. \right\} 2^2 = 4 \quad \text{for 8 slots} \Rightarrow 2^8 = 256$$

\Rightarrow character is also represented as number internally.

ASCII \rightarrow A \rightarrow 65, a \rightarrow 97, 0 \rightarrow

8bit char

$\underbrace{_ _ _ _ _ _ _ _}$, use

use first bit to represent sign

+0
-0
are same

\Rightarrow signed basically means you have both signed & negative & positive values.

\Rightarrow unsigned - only positive values

- ⇒ **short** means you are going to use lesser amount of memory.
- ⇒ **long** means larger memory.
- ⇒ **float** can store
- ⇒ **long double** - for storing extremely large numbers use for scientific purpose.

main ()

```

{
    printf ("sizeof (char) = %.1f byte \n", sizeof (char));
    printf ("sizeof (short) = %.1f byte \n", sizeof (short));
    printf ("sizeof (int) = %.1f byte \n", sizeof (int));
    printf ("sizeof (long) = %.1f byte \n", sizeof (long));
    printf ("sizeof (double) = %.1f byte \n", sizeof (double));
    printf ("sizeof (float) = %.1f byte \n", sizeof (float));
    printf ("sizeof (long double) = %.1f byte \n", sizeof (long double));
}
return 0;
}

```

Sizeof() : this function is used for getting size of what you give

Phone: +91 844-844-0102

47

find factorial of large no. 1000 and store it.

1000! → linked-list

↳ Integer can not store such a large no.

[3.4] Variables

int a;



creating memory for storing some integer here a is called variable.

all variable names → Identifiers

↳ Data types

main()

int a;

a = 10;
a = 5;

}

here value is changing from 10 to 5
value is varied to its called variable

2 Main concept of variable

→ Declaration

→ Initialization

→ Usage

⇒ you have to always declare a variable before using it

Mail: gatecse@appliedroots.com

⇒ Typically variable name starts with **Name** with lower case 0 to 9 and underscore.

fName

⇒ We give variable names, which are easy to read.
int id; int x;

float f = 0.23;

double d = 1.2e7 Scientific representation
 1.2×10^7

⇒ int a, b, c, d; declaring multiple variable in a single line

⇒ int a=5, b=5; declaring & initializing variables

⇒ Put $\underbrace{a, b, c}_{\text{declaration}} = \underbrace{0}_{\text{initialization}};$ three variables with value 0

int a;
int b;
int c;
a=0;
b=0;
c=0;

Instead of writing 6 line code, we are writing it in one line.

3.5 Constants

→ do not change

Phone: +91 844-844-0102



Types: char, string, integer, real-valued

define constant : const int a=5;

char : '#', 'j', '\$'

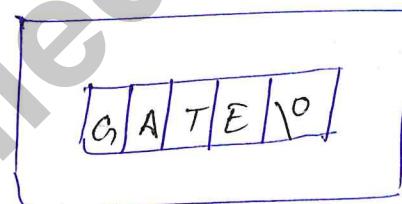
'abc' X Not a char

'' X empty can not be char

'q' X Not valid char w/o single quotes.

String : sequence of characters.

"GATE"



$10 - \text{ASCII} = 0$

Null terminator

⇒ If we don't store back-slash (\0) then it will everything it gets because memory is contiguous.

⇒ (\0) for this, we get to know our string is ended here.

Real-value : 1.2F, 1.2f, 1.2L, 1.2l

Integer \rightarrow decimal system Phone: +91 844-844-0102



Const int a = 123;

$$\begin{array}{r} 1 \quad 2 \quad 3 \\ \times \quad \times \quad \times \\ 10^2 \quad 10^1 \quad 10^0 \\ \hline 100 + 20 + 3 = 123 \end{array}$$

123456L

↳ long int const

1234U

↳ unsigned integer const

1234UL

↳ unsigned long integer constant

Octal System \rightarrow 0, 1, 2, 3, 4, 5, 6, 7 (only 8-digits)
(Base 8)

int a = 012; { read this as octal as we are placing 0
 ^
 octal number

$$\begin{array}{r} 1 \quad 2 \\ \times \quad \times \\ 8^2 \quad 8^0 \\ \hline \end{array}$$

$$8 + 2 = 10 \text{ decimal}$$

Hexa-decimal (Base 16)

0, 1, 2 - 9, A, B, C, D, E, F
 |
 10
 |
 15

0x or 0xe

0 followed by capital or small x. int a = 0x12

Mail: gatecse@appliedroots.com

(12)₁₆

$$\begin{array}{r}
 1 \quad 2 \\
 \times \quad \times \\
 16^2 \quad 16^0 \\
 \hline
 16 + 2 = 18
 \end{array}$$

int a = 8, 126; **Phone: +91 844-844-0102**

2.5 ✗ decimal
12.5 ✗ space

main

⇒ Other way of defining constants **#define**

#define PI 3.14159

main ()

{

const a=5;

printf("PI: .f", PI);

int x=012;

printf("In x=%d", x);

int y=0x12;

printf("In y=%d", y);

}

⇒ Once you have a constant, you can't change it during the program.

O/P:

PI = 3.14159

x = 10

y = 18

$$012 = (12)_8 = (10)_{10}$$

$$0x12 = (12)_{16} = (18)_{10}$$

K & R → C Concise & crisp
in depth → examples (Shrivastava)
cover most cases

[3.6] Input and output

printf, scanf are two popular function c.

↓ + ↗
o/p + formatted

#include <stdio.h> → Print, Scan are defined in stdio.h file

main()

(standard input output)

{
 char ch = 'a'; → a
 char str[10] = "abcdef";
 float real = 12.34;
 int number = 100;
 double dbl = 12.345678;

a/b/c/d/e/f\n

%u → unsigned integer
%f → floating number

gcc compiler includes stdio.h automatically.

printf ("Character: %c\n", ch);

printf ("Character as integer: %d\n", ch);

printf ("String: %s\n", str);

printf ("float: %f\n", real);

printf ("Double: %lf\n", dbl); long float

printf ("Scientific (e): %e\n", dbl);

printf ("Scientific (E): %E\n", dbl);

printf ("Integer: %d\n", number);

printf ("Octal: %o\n", number);

printf ("Hexadecimal: %x\n", number);

printf ("Hexadecimal: %X\n", number);

number, str, real);

Float: %f\n",
Multiple spaces

O/P:
Character: a
Character as int: 97
String: abcdef
Float: 12.340000

Hexadecimal: 64 String: abcdef
Float: 12.34000000

Double: 12.345678
Scientific (P): 1.234568 e+01
Scientific (E): 1.234568 E+01

Integer: 100
Octal: 144

Hexadecimal: 64 } doesn't addl 0x Prefixes

Mail: gatecse@appliedroots.com

[3.7] Formatting input and output

Phone: +91 844-844-0102

25

Integers:

`scanf("%2d %3d", &a, &b);`

① 6, 39 → $\boxed{6}$ $\boxed{39}$

② 12 123 → a → 12 b → 123

③ 123 1234 → a → 12 b → 3

④ 12 1234 → a → 12 b → 123 when it get space
it think, I am done

`%n%d` → Where n represents no. of integers, it will read

`printf("a = %4d, b = %3d", a, b);` → formatted printing

① $a = 12, b = 14$

$a = -12, b = -14$

Floating point numbers

`scanf("%2f %4f", &x, &y);` ← Input

① 5 5.9 $x = 5.0$ $y = 5.9$

② 5.1 1.23 $x = 5.1$ $y = 1.23$

③ 1.23 4.5678 $x = 1.2$ $y = 3.0$

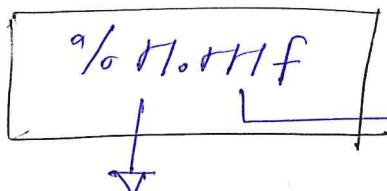
Printf (%.4lf, y = %.2f); **844-844-0102**

APPLIED
ROOTS

Print only 1 digit after decimal prints

① $x = 8.0, y = 5.9 \Rightarrow x = -8.0, y = -5.9$

② $x = 15.231, y = 65.12345 \Rightarrow x = 15.2, y = -65.12$



→ no. of digits after the decimal

Min. no. of chars

String

char str[10];

Scarf ("%s", str);

i/p - abcdef

Scarf ("%3s", str);

abcdef\0

i/p : abcdef

Printf ("%3s", "abcdef"); Str : abc\0 ignore def

→ abcdef

* Printf ("%5s", "ab"); → --- ab

Printf ("%3s", "abcdef"); → abc

→ In string also, you can follow `%n.nlf` format

Suppression char

don't take second integer

Mail: gatcse@appliedroots.com

25 30 35

Scanf ("%1d%*d%1d", &a, &b, &c)

a = 25 b = 35 (random garbage)

Printf scanf

Phone: +91 844-844-0102

⇒ getchar() and putchar()
I/O

are available for character

char c;

Printf ("enter a@ char");

c = getchar();

behaves same as Scanf("a,c", &c);

Bindf ("The value you entered"),

Putchar (c);

[4] Operators and Expressions

⇒ operations on integers, floats and logical

[4.1] Arithmetic operators : Integers

/* ----- */ Comment line

⇒ Whatever is written in between
this is totally ignored.

⇒ Multiline comment

⇒ Comments are english sentences,
which we write for our clarity

// ----- ⇒ Single line comment

Unary operators : $HNP \rightarrow 1$

-3 here - is unary operator

Binary operators : $B_1 \rightarrow 2$

Mail: gate@appliedroots.com

int main()

```

int x=-3, y=+4;           // unary integers
printf("x.d\|t y.d\|n", x, y);
int a=17, b=4;
printf("Sum = %d\n", a+b);
printf("Difference = %d\n", a-b);
printf("Product = %d\n", a*b);
printf("Quotient = %d\n", a/b);
printf("Remainder = %d\n", a%b);
return 0;
}

```

$$\text{Sum} = 21$$

$$\text{Difference} = 13$$

$$\text{Product} = 68$$

$$\text{Quotient} = 4$$

$$\text{Remainder} = 1$$

$$\frac{17}{4} = 4.25 \quad \text{take 4 only}$$

[4.2] Arithmetic operations : floats

⇒ Modulo is not meaningful when you are dividing real numbers. It is meaningful when you are dividing an integer with an integer.

int main()

```

float a=13.4, b=3.2;
printf("Sum is %.2f\n", a+b);      16.60
printf("Difference = %.2f\n", a-b);  10.2
printf("Product = %.2f\n", a*b);    42.88
printf("a/b = %.2f\n", a/b);       4.19

```

```
int x=7, y=2;
float z=20;
```

```
printf("In 7/2 : %f", x/y); // int/int is an int
printf("In 7/2 : %d", x/y);
printf("In 7/2.0 : %f", x/z); // (int/float) or (float/int)
                             // is a float
return 0;
}
```

$7/2 = 4.18$

$7/2 = 3$

$7/2 = 3.500,000$

[4.3] Assignment operators

```
#include <stdio.h>
int main()
{
    int x,y,z,t;
    x=5;
    y=6;
    z=x+y*z;
    t=x; t=5
    printf("In x=%d, y=%d, z=%d, t=%d", x,y,z,t);
}
```

O/P : $x=5, y=6, z=17, t=5$
 $x=0, y=50, z=5, t=5$

$x=y=z=10$; // Multiple variables with same value.
 $x=x+5$; $x+=5$; $y*=5$; $z*=2$; $x*=5$;
 printf("In x=%d, y=%d, z=%d, t=%d", x,y,z,t);
 return 0;

[4.4] Increment & Decrement operators

Phone: 844-0102

⇒ There are two types of operators

- Prefix operators.
- Postfix operators.

int main()

{

 int x=8;

- ① `printf("x=%d\n", x);`
- ② `printf("x=%d\n", ++x);` // prefix increment
- ③ `printf("x=%d\n", x);`
- ④ `printf("x=%d\n", --x);` // prefix decrement
- ⑤ `printf("x=%d\n", x);`
 return 0;

O/P: x=8 x=9 x=9 x=8 x=8

for postfix operators:

- ① use the variable x
- ② $x = x + l$

② and ④ statement as postfix then

O/P: x=8 x=8 x=9 x=9 x=8

#include <stdio.h>

int main ()

{

int m=10;

int n, n1;

n = ++m;

n1 = m++;

n--;

--n1;

n = n1;

printf ("%d", n);

return 0;

}

$$m = 10 \text{ II}$$

$$n = ++m = ++10 = 11$$

$$n_1 = m++ = 11++ = 11$$

$$m = \cancel{11} \text{ II}$$

$$n-- = 11-- = 10$$

$$--n_1 = --11 = 10$$

$$n = n - n_1$$

$$= 10 - 10$$

$$= 0$$

The output of the code is 0

[4.6] Relational operators

<, >, <=, >=, are relational operators.

#include <stdio.h>

int main ()

{

int a, b;

scanf ("%d %d", &a, &b);

→ 12, 4

printf ("enter values for a & b:");

if (a < b)

printf ("val is less than %d\n", a);

if ($a \leq b$)

Phone: +91 844-844-0102



printf ("%.d is less than or equal to %.d\n", a, b);

if ($a == b$)

printf ("%.d is equal to %.d\n", a, b);

if ($a > b$)

printf ("%.d is greater than %.d\n", a, b);

if ($a >= b$)

printf ("%.d is greater than or equal to %.d\n", a, b);

if ($a != b$)

printf ("%.d is not equal to %.d\n", a, b);

if ($a == b$) // value of expression is same as the value

of b and not 1 or 0.

$a = 5$
 $a = 4$

printf ("Assignment operator");

printf ("%.d", a == b);

return 0;

}

O/P:

enter values for a & b : 12 4

12 is not equal to 4

12 is greater than 4

12 is greater than or equal to 4

4

[4.7] Logical or Boolean operators

Phone: +91 844-844-0102

52

→ logical or boolean operators $\&\&$ AND
 $!!$ OR

```
int main()
{
    int a=40, b=20;
    int c=20, d=30;
    if (a>b  $\&\&$  a!=0)           40 > 20
    {
        printf ("&& operators: Both conditions are true\n");
    }

    if (c>d  $!!$  d!=20)
    {
        printf ("|| operator: only one condition is true\n");
    }

    if (! (a>b  $\&\&$  a!=0))
    {
        printf ("! operator: Both conditions are true\n");
    }
    else
    {
        printf ("! operator: Both conditions are true, But
                status is inverted or false\n");
    }

    if (a  $\&\&$  b){ printf ("In a  $\&\&$  b); }
    if (a  $\&\&$  0){ printf ("In a  $\&\&$  0); }
    if (a  $!!$  0){ printf ("In a  $!!$  0); }
}
```

Mail: gatcse@appliedroots.com

```
#include <stdio.h>
int main()
{
    int a, b, c, sum;
    sum = (a=8, b=7, c=9, a+b+c);
    printf("sum=%d\n", sum);
    return 0;
}
```

O/P => 24

⇒ Works left to right

$$\begin{aligned} \text{Sum} &= (a=8, b=7, c=9, a+b+c) \\ a \rightarrow 8 & \quad \downarrow \quad \downarrow \quad \downarrow \\ b \rightarrow 7 & \quad 8 \quad 7 \quad 9 = 24 \\ c \rightarrow 9 & \end{aligned}$$

⇒ returns value of right most sub-expression.

```
#include <stdio.h>
int main()
{
    int a=8, b=7, temp;
    printf("a=%d, b=%d\n", a, b);
    temp = a, a=b, b=temp;
    printf("a=%d, b=%d\n", a, b);
    return 0;
}
```

$$\begin{aligned} \text{temp} &= 8 \\ a &= 7 \\ b &= 8 \end{aligned}$$

[4.9] Condition (ternary) Operator

31

→ Ternary means something that contains three parts.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int a, b, max;
```

```
    printf("Enter values for a and b:");
```

```
    scanf("%d %d", &a, &b);
```

```
    max = a > b ? a : b;
```

```
    printf("Larger of %d and %d is %d\n", a, b, max);
```

```
    return 0;
```

```
}
```

O/P: $a \rightarrow 4, b \rightarrow 2$

$a > b ? a : b$

$4 > 2 \quad (4)$

If a is greater than b then returns a else b

⇒ This ternary operator can also be written using the following code.

```
if (a > b)
```

```
{
```

```
    printf("a is greater);
```

```
}
```

```
else {
```

```
    printf("b is greater);
```

[4.10] Solved Problem Gate 2008

Photo 91 844-844-0102



Q: Which combination of the integer variables x , y and z makes the variable a get the value 4 in the following expression

$$a = (x > 4) ? ((x > z) ? x : z) : ((x > z) ? 4 : z)$$

(A) $x=3, y=4, z=2$

(B) $x=6, y=5, z=3$

(C) $x=6, y=3, z=5$

(D) $x=5, y=4, z=5$

Solution:

$$a=4$$

Nested ternary

$$(x > 4) ? ((x > z) ? x : z) : ((x > z) ? 4 : z)$$

(A) $x > 4$ goto false

$$x > 2 \quad \checkmark \quad 4 = 4$$

so option (A) is correct.

[4.11] sizeof operator

⇒ sizeof is an unary operator. It only takes one parameter.

⇒ operates on single parameter. That's why it is called unary operator

Mail: gatocse@appliedroots.com

1) `printf("%d", sizeof(int));` Phone: +91 844-844-0102

\uparrow
datatype

No. of bytes are
are required to
store an integer

`int a;`

2) `printf("%d", sizeof(a));`

\uparrow variable

3) `printf("%d", sizeof(2));`

\uparrow constant

\Rightarrow `sizeof()` is not a function. It's an operator which is used in C.

\Rightarrow It is used to get no. of bytes it takes to make your code portable

[4.12] Precedence & Associativity

`int a = 2 * 3 + 5`

possibilities:

$$(2 * 3) + 5 = 11$$

$$2 * (3 + 5) = 16$$

\Rightarrow C says multiplication must be performed before addition.

associativity: have two operators of same precedence.

(L to R)

Mail: gateteacher@appliedroots.com

$$100 / 100 = 1$$

	Operators	Associativity	Precedence
{ () [] . →	Parenthesis to function call Array subscript Dot (Member of structure) Arrow (Member of structure)	Left to right	Highest 14
! - + - ++ -- & * (+type) sizeof	Logical NOT One's complement Unary Minus(Negation) Increment Decrement Address of Indirection Cast Sizeof	Right to left	13
* / %	Multiplication Division Modulo (Remainder)	Left to right	12
+ -	Addition Subtraction	Left to right	11
<< >>	Left shift Right shift	Left to right	10
<= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	8
!=	Equal to NOT equal to	Left to right	8

	Operators	Associativity	Precedence
&	Bitwise AND	Left to Right	7
^	Bitwise XOR	"	6
	Bitwise OR	"	5
&&	Logical AND	"	4
	Logical OR	"	3
? :	Conditional	Right to Left	2
=, += *=, etc	Assignment operators	Right to left	1
,	Comma	Left to right	Lowest 0

PUMA'S REBL TAC

⇒ Unary, conditional (ternary), and assignment operators are right to left.

* Auto devices are always right

AUT [Assignment, Unary, Ternary operators]

```
int main()
{
    int a, b, c, d, e, f, g, h, k;
    a=8, b=4, c=2, d=1, e=5, f=20;
    printf ("%d\n", a+b-(c+d)*3%e+f/g);
```

$$8+4 - (2+1) * 3 \% 5 + 20/9$$

$$8+4 - 3 * 3 \% 5 + 20/9$$

$$8+4 - 3 * 3 \% 5 + 20/9$$

$$8+4 - 3 * 3 \% 5 + 20/9 = 10$$

a = 17, b = 5, c = 6, d = 3, e = 5, phone: +91 844-844-0102

*Printf("%d\n", a * 6 - b / 2 + (c * d - 5) / e);*

$$17 * 6 - 5 / 2 + (6 * 3 - 5) / 5$$

$$17 * 6 - 5 / 2 + 13 / 5$$

$$= 5 - 2 + 2$$

$$= 5$$

a = 4, b = 5, c = 6, d = 3, e = 5, f = 10;

*Printf("%d\n", a * b - c / d < e + f);*

$$4 * 5 - 6 / 3 < 5 + 10$$

$$20 - 2 < 15$$

$$18 < 15 \quad \text{false} \quad 0$$

a = 8, b = 5, c = 8, d = 3, e = 65, f = 10, g = 2;

h = 5, k = 2;

*Printf("%d\n", a - b + c / d == e / f - g + h * k);*

$$8 - 5 + 8 / 3 == 65 / 10 - 2 + 5 * 2$$

$$8 - 5 + 2 == 6 - 2 + 1$$

$$5 == 5 \quad \text{true} = 1$$

a = 8, b = 3, c = 2, d = 3, e = 2, f = 11

*Printf("%d\n", a - b || (a - b * c) + d && e - f % 3);*

&& has higher precedence than ||.

$$8 - 3 || (8 - 3 * 2) + 3 \&\& 2 - 11 \% 3$$

$$8 - 3 || 2 + 3 \&\& 2 - 2$$

$$5 || 5 \&\& 0 = 5 || 0 = 1 = \text{true}$$

Mail: gatecse@appliedroots.com

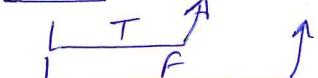
$$\textcircled{1} \quad y = (+x) + (-x)$$

C does not clearly specify
which operand should be
executed first.
Phone: +91 844844-0702

do not write code like this

tertiary

$$\textcircled{2} \quad a > b ? a : b$$



$$\textcircled{3} \quad () \& () \& () \quad \{ \text{False}$$

\textcircled{1}

\textcircled{2}

\textcircled{3}

evaluated



If this is false

Whole expression will not be evaluated at all.

$$\textcircled{4} \quad (10 > 2) || () \quad \{ \text{True}$$



If this is true

Whole expression will not be evaluated at all

If this is false \rightarrow this will be evaluated, If this is true then it will also be true

[4.13] Type Casting / conversion

$$\text{float } f = 2.0 + 3$$

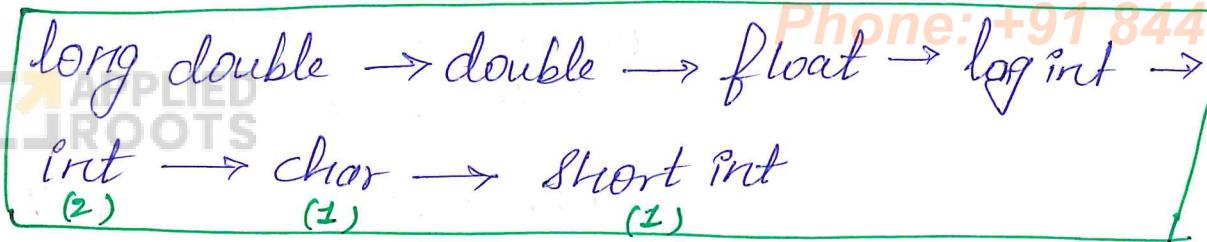


float constant

$$\textcircled{1} \quad 3 \rightarrow 3.0$$

$$\textcircled{2} \quad 2.0 + 3.0 = 5.0$$

$$\textcircled{3} \quad f = 5.0$$



\Rightarrow Suppose long int is added to int.

$$= 2L + 3 \rightarrow 3L$$

$$= 2L + 3L$$

$$= 5L$$

\Rightarrow What if we have unsigned int

$$= \text{unsigned int} + \text{long int}$$

= $\underbrace{\text{try to convert into}}$

Case 1:

If unsigned int fits
into long int

Case 2:

If unsigned int is a
large number, both of
them will be converted
into unsigned long int.

How type conversion happens during assignment operation

Ex: 1

$$\text{int } i = 80.56;$$

We are trying to store float
into integer.

Automatic promotion

Ex: 2

$$\text{char } c_1 = 'l'; \rightarrow \text{ASCII value to } 80 = P$$

Ex: 3

$$\text{float } f_1 = 12.6$$

$$\text{int } i_1 = f_1; \quad i_1 \rightarrow 12$$

$$\text{float } f_2 = i_1; \quad f_2 \rightarrow 12.0$$

$$\text{Mail: gatecse@appliedroots.com}$$

⇒ If you want to write reliable code, better to use explicit type cast. Phone: +91 844-844-0102

It's unary operator

Ex:

int $x = 20, y = 3;$

float $z = x/y; \quad // z = 20/3 \rightarrow 6.0 \quad \} \text{automatic type cast}$

$\frac{x}{y}$

float $z = (\text{float}) x/y;$

$$\begin{aligned} 20.0/3 &= 20.0/3.0 \\ &= 6.66 \end{aligned}$$

float $z = (\text{float})(x/y)$

division will happen first as parenthesis has higher precedence than type cast.

[4.14] Bitwise operations

⇒ Binary - 0, 1 (base 2) Decimal $\rightarrow 0 \dots 9$ (base 10)

⇒ At the end every computer uses 0, 1.

⇒ 8-bits = 1 byte

$$\begin{array}{r} 2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\ \hline \end{array}$$

bits $\rightarrow 0/1$

$$(20)_{10} = (\quad)$$

$$20 \rightarrow 16 + 4$$

$$(00010100)_2 \rightarrow \begin{pmatrix} \text{binary representation} \\ \text{of 20} \end{pmatrix}$$

Mail: gatecse@appliedroots.com

Phone: +91 844-844-0102

$$(10)_{10} = \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{23} \underline{24}$$

**APPLIED
ROOTS**

$$\begin{array}{r} 8+2 \\ + \\ \hline 23+24 \end{array}$$

Bitwise AND (&)

$$a = 20$$

$$b = 10$$

$$a \& b = \begin{array}{r} 0001,0100 \\ 00001010 \\ \hline \end{array}$$

$$(0)_{10} = \underline{(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)}_2$$

$$20 = (0001,0100)_2$$

$$10 = (00001010)_2$$

0 → false
1 → true

Bitwise OR (|)

$$a | b = \begin{array}{r} 00010100 \\ 00001010 \\ \hline 00011110 \\ 24-23-22-21-20 \end{array} \rightarrow (30)_{10}$$

Bitwise XOR

x	y	x xor y
0	0	0
0	1	1
1	0	1
1	1	0

Mail: gatecse@appliedroots.com

a^nb^n

$$\begin{array}{r}
 00010100 \\
 00001010 \\
 \hline
 00011110
 \end{array}$$

Left shift operation

Left shift ($<<$)

$a << 2$ Shift all the bits left by 2

$$(00010100) << 2$$

$$\begin{array}{r}
 01010000 \\
 \downarrow 2^4 \\
 2^6
 \end{array} = 64 + 16 = 80$$

\Rightarrow left shift by 2-bits means add two 0's at the end and discard first two digits.

$$\begin{array}{c}
 2 \text{ Bytes} \\
 (1100,0000,00010100) << 2
 \end{array}$$

$$0000,0000,01010000$$

Right shift ($>>$)

$a >> 4$ Shift all bits right by 4

$$\begin{array}{r}
 00010100 \\
 \xrightarrow{\quad} \\
 >> 4
 \end{array}$$

$$\underline{000000010100}$$

→ To change the order of execution.

if--else, for loops,

Basic Control Statement

[6.1] If else

```
#include <stdio.h>
int main()
{
    int a, b;
    printf("enter two numbers:");
    scanf("%d %d", &a, &b);
    if(a>b)
        printf("Bigger number=%d\n", a);
    else
        printf("Bigger number = %d\n", b);
    return 0;
}
```

O/P → a → 12 , b → 4

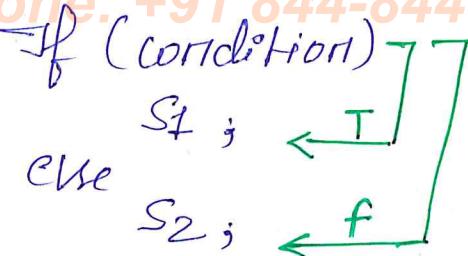
$$12 > 4$$

Bigger number = 12

if (condition)

S₁ ;

Phone: +91 844-844-0102



⇒ We can have if without else.

Program to check entered number is negative.

#include <stdio.h>

{

int num;

printf ("enter a number:");

scanf ("%d", &num);

if (num < 0)

{

printf ("Number entered is negative\n");

num = -num;

}

printf ("value of num is : %d\n", num);

return 0;

}

} block

O/P:

entered a number: -56
Number entered is negative
value of num is : 56

Program to find whether a year is leap or not **Phone: +91 844-844-0102**



```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int year;
```

```
    printf("Enter Year:");
```

```
    scanf("%d", &year);
```

```
    if (year % 100 != 0)
```

```
{
```

```
        if (year % 4 == 0)
```

```
            printf("Leap Year\n");
```

```
        else
```

```
            printf("NOT leap\n");
```

```
}
```

```
    else
```

```
{
```

```
        if (year % 400 == 0)
```

```
            printf("Leap Year\n");
```

```
        else
```

```
            printf("NOT leap\n");
```

```
}
```

1900 → else

1900 % 400

```
2000 → else → 2000 % 400
```

↳
leap year

↳
Not leap year

Mail: gatecse@appliedroots.com

#WAP to find out grade of a student when the marks of 4-subjects are given.

Phone: +91 844-844-0102

$$P >= 85 \quad g = A$$

$$P < 85 \text{ and } P >= 70 \Rightarrow g = B$$

$$P < 70 \text{ and } P >= 55 \Rightarrow g = C$$

$$P < 55 \text{ and } P >= 40 \Rightarrow g = D$$

$$P < 40 \Rightarrow g = E$$

#include <stdio.h>

int main()

{

float m1, m2, m3, m4, total, Per;

char g;

printf("enter marks of 4 subjects:");

scanf("%f %f %f %f", &m1, &m2, &m3, &m4);

total = m1 + m2 + m3 + m4;

Per = total / 4;

if (Per >= 85)

g = 'A';

else if (Per >= 70)

g = 'B';

else if (Per >= 55)

g = 'C';

else if (Per >= 40)

g = 'D';

else

g = 'E';

return 0;

printf("Percentage is %.f, Grade is %.1c\n", Per, g);

Mail: gntocse@appliedroots.com

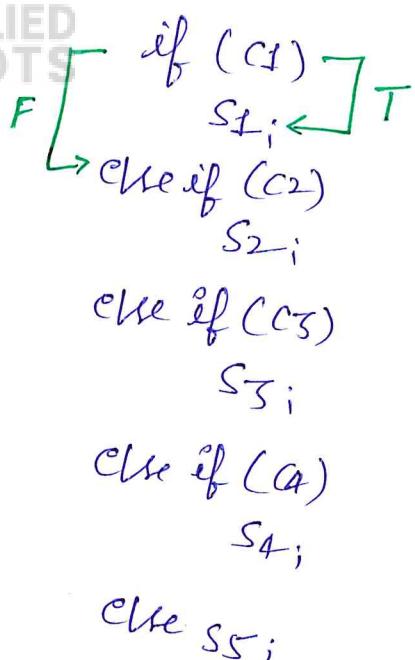
0/p:

enter marks of 4 subjects: 45 55 65

Percentage is 60.000000, Grade is C

10
If else ladder:

Phone: +91 844-844-0102



[6.2] while loop

⇒ Works same as while loop & for loop.

WAP to print numbers from 1 to 10 using while loop

```
#include <stdio.h>
int main()
{
    int i=1;
    while (i<=10)
    {
        printf ("%d\n", i);
        i=i+1;
    }
    printf ("\n");
    return 0;
}
```

⇒ When there is repetition then we can use loops.

Phone: +91 844-844-0102

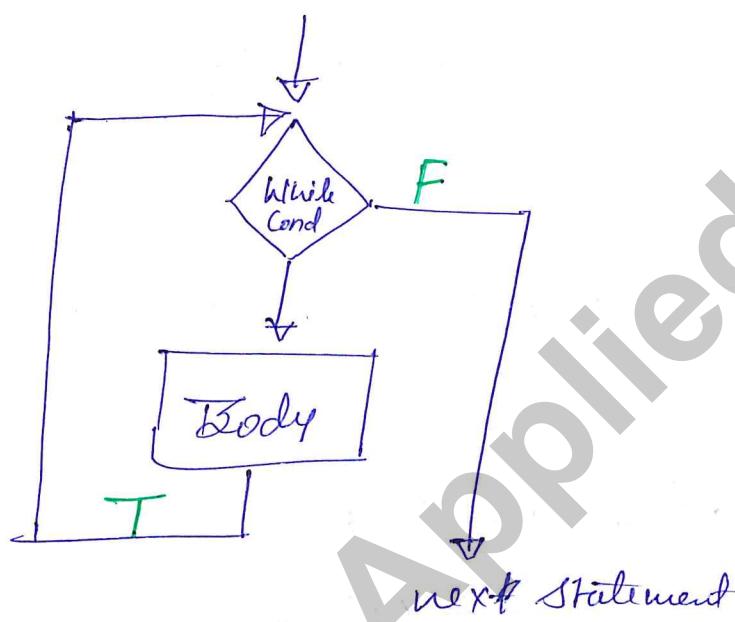
While (Condition)

```
{  
    S1;  
    S2;  
    S3;  
    S4;  
}
```

While (Condition)

```
S1;
```

Pre - Statement (outside)



WAP to print the sum of digits of any number.

```
int main ()  
{  
    int n, sum = 0, rem;  
    printf ("enter a number:");  
    scanf ("%d", &n);  
    while (n>0){  
        rem = n % 10;  
        sum += rem;  
        n / = 10;  
    }  
    printf ("sum of digits = %d\n", sum);  
    return 0;
```

O/P :

enter a number: 1234
Sum of digits = 10

Mail: gatecse@appliedroots.com

$n = 1234$

Phone: +91 844-844-0102



While ($1234 > 0$) ✓

$$rem = 1234 \% 10 = 4$$

$$Sum = Sum + rem = 0 + 4 = 4$$

$$n = n / 10 = 1234 / 10 = 123$$

(2) While ($123 > 0$)

$$rem = 123 \% 10 = 3..$$

$$Sum = 4 + 3 = 7$$

$$n = 123 / 10 = 12$$

∴ Sum of digits = 10

WAP to convert a binary number into decimal number.

int main()

{

int n, binary, rem, d, j=1, dec=0;

Printf("enter the number in binary:");
Scanf("%d", &n);

binary = n;

While ($n > 0$)

{

rem = n % 10;

d = rem * j;

dec + = d;

j * = 2;

n / = 10;

}

Printf("Binary number = %d, Decimal number = %d\n", binary, dec);
return 0;

(3) While ($12 > 0$)

$$rem = 12 \% 10 = 2$$

$$Sum = 7 + 2 = 9$$

$$n = 12 / 10 = 1$$

(4) While ($1 > 0$)

$$rem = 1 \% 10 = 1$$

$$Sum = 9 + 1 = 10$$

$$n = 1 / 10 = 0$$

While ($0 > 0$) ✗

Enter the number in binary: 10101

Phone: +91 844-844-0102



(10101 > 0) ✓

② while (1010 > 0)

$$\text{rem} = 10101 \% 10 = 1$$

$$\text{rem} = 1010 \% 10 = 0$$

$$d = 1 * 1 = 1$$

$$d = 0 * 1 = 0$$

$$\text{dec} = \text{dec} + d = 0 + 1 = 1$$

$$\text{dec} = 1 + 0 = 1$$

$$f = f * 2 = 2$$

$$f = 2 * 2 = 4$$

$$n = 10101 / 10 = 1010$$

$$n = 1010 / 10 = 101$$

③ while (101 > 0)

$$\text{rem} = 101 \% 10 = 1$$

④ while (10 > 0)

$$\text{rem} = 10 \% 10 = 0$$

$$d = 1 * 1 = 1$$

$$d = 0 * 1 = 0$$

$$\text{dec} = \text{dec} + d = 1 + 1 = 2$$

$$\text{dec} = 0 + 5 = 5$$

$$f = 2 * 1 = 2$$

$$f = 2 * 2 = 4$$

$$n = 101 / 10 = 10$$

$$n = 10 / 10 = 1$$

⑤ while (1 > 0)

$$\text{rem} = 1 \% 10 = 1$$

$$d = 1 * 1 = 1$$

$$\text{dec} = 5 + 1 = 6$$

$$f = 2 * 1 = 2$$

$$n = 1 / 10 = 0$$

Binary number = 10101, Decimal number = 21

[6:3] do-while loop

Phone: +91 844-844-0102



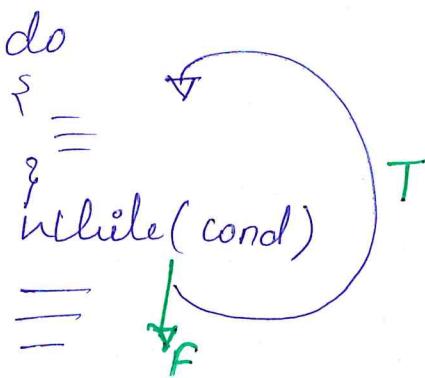
while (cond)

{
=

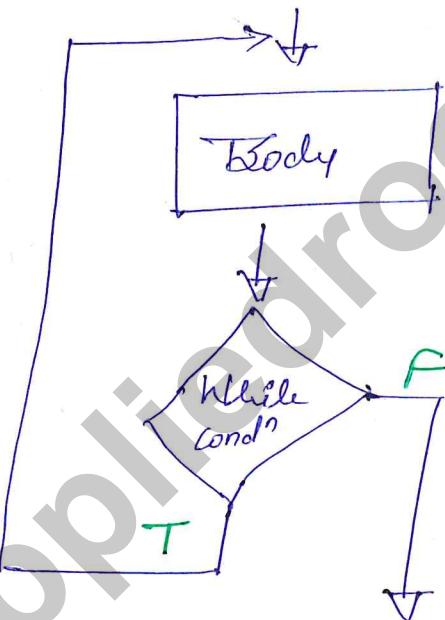
do

?
S1;

while (cond)



Brv - Stmt (outside)



⇒ Body of do-while loop will be executed atleast once.

```
int main ()  
{  
    int i = 11;  
    do  
    {  
        printf(“.1.d\t”, i);  
        i = i + 1;  
    } while (i <= 10)  
    printf(“\n”);  
    return 0;
```

O/P = 11

for (exp₁; exp₂; exp₃)
 ——————
 init condition ↳ update { ≡ }

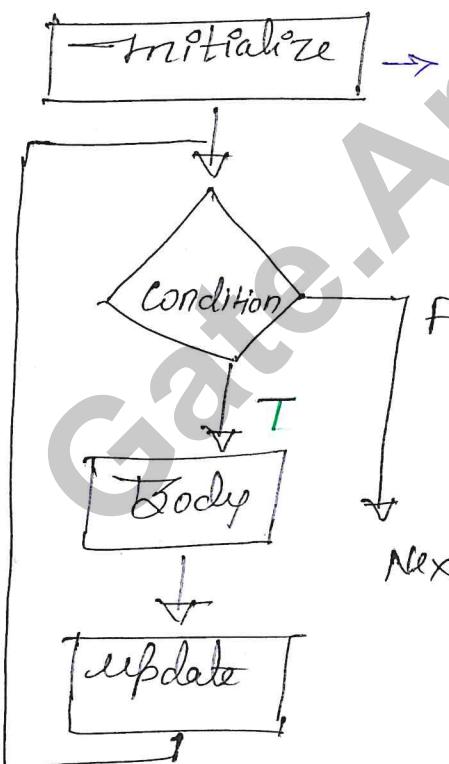
#include <stdio.h>

```
{
  int k;
  for(k=10; k>=2; k=2)
    printf("%d", k);
    printf("\n");
  return 0;
}
```

⇒ In body part,
 either we can have
 one statement or
 multiple statements.

$k = 10$ exp₁
 $10 \geq 2$ ✓ exp₂
 $k = 10$
 $k = k - 2 = 10 - 2 = 8$ exp₃

Prev - Start (outside)



$B \geq 2$	$4 \geq 2$
$K = 8$	$K = 4$
$K = 6$	$K = 2$
$G \geq 2$	$2 \geq 2$
$K = 6$	$K = 2$
$K = 4$	$K = 0$
	$O \geq 0$

for (init; cond; update)

{
 body
 }

↔

Print
 While (cond) {
 Body
 Update
 }

run a loop n-times **Phone: +91 844-844-0102**

```
int i;  
for (i=0; i<n; i++) {  
    }  
    } n times  
    }  
    i = 0, i = 1, i = 2, ..., i = n-1
```

find the sum of this series upto n terms.

$$1+2+4+7+11+16+\dots$$

```
#include <stdio.h>
```

```
int main()
```

```
{  
    int n, i, sum = 0, term = 1;  
    printf("enter number of terms : ");  
    scanf("%d", &n);  
    for (i=1; i<=n; i++)  
    {  
        sum += term;  
        term = term + i;  
    }  
    printf("In %d", term);  
}
```

```
printf("The sum of series upto %d terms is %d",  
n, sum);
```

```
return 0;  
}
```

O/P : enter number of terms : 5

Mail: gatetcse@appliedroots.com

Term	1st	2nd	3rd	4th	5th
term	1	2	4	7	11

$$i=1 \quad 1 <= 5$$

Phone: +91 844-844-0102

$$\text{Sum} = \text{Sum} + \text{term} = 0 + 1 = 1$$

$$\text{term} = \text{term} + i = 1 + 1 = 2$$

$$\text{Pf(term)} = 2$$

$$\text{Update } i \Rightarrow i++ = 1+1 = 2$$

$$2 <= 5$$

$$\text{Sum} = 1 + 2 = 3$$

$$\text{term} = 2 + 2 = 4$$

$$\text{Pf(term)} = 4$$

$$i=3$$

$$3 <= 5$$

$$\text{Sum} = 3 + 4 = 7$$

$$\text{term} = 4 + 3 = 7$$

$$\text{Pf(term)} = 7$$

$$i=4$$

$$4 <= 5$$

$$\text{Sum} = 7 + 7 = 14$$

$$\text{term} = 7 + 4 = 11$$

$$\text{Pf(term)} = 11$$

$$i=5$$

$$5 <= 5$$

$$\text{Sum} = 14 + 11 = 25$$

$$\text{term} = 11 + 5 = 16$$

$$\text{Pf(term)} = 16$$

$$i=6$$

$$6 <= 5 X$$

The sum of series upto 5 terms is 25.

Fibonacci numbers

⇒ Fibonacci sequence appears in Indian mathematics in connection with Sanskrit prosody as pointed out by Parmanand Singh in 1985.

⇒ F_n denotes term of sequence.

$$F_0 = 0, F_1 = 1 \quad \text{and} \quad F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 1$$

Mail: gatecse@appliedroots.com

#WAP to print fibonacci series

Phone: +91 844-844-0102

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

(In this series each number is a sum of the previous two numbers)

#include < stdio.h >

int main ()

{

long x, y, z;

int i, n;

x = 0; y = 1;

printf ("enter the number of terms: ");

scanf ("%d", &n);

printf ("%ld", y);

for (i = 1; i < n; i++)

{

z = x + y

printf ("%ld", z);

x = y;

y = z;

}

printf ("\n");

return 0;

}

$$x = 0, \boxed{y = 1}$$

enter the number of terms: 5

$$n = 5$$

$$y = 1$$

$$i = 1, 1 < 5$$

$$z = x + y = 0 + 1 = 1$$

$$x = 1, y = 1$$

$$i = 2, 2 < 5$$

$$z = 1 + 1 = 2$$

$$x = 1, y = 2$$

$$i = 3, 3 < 5$$

$$z = 1 + 2 = 3$$

$$x = 2, y = 3$$

$$i = 4, 4 < 5$$

$$z = 2 + 3 = 5$$

$$x = 3, y = 5$$

$$i = 5, 5 < 5$$

X

Mail: gatecse@appliedroots.com

⇒ We can use comma operator with int for loop.

Phone: +91 844-844-0102

54

#include <stdio.h>

int main ()

{

int i, j;

for (i=0, j=10 ; i<=j ; i++, j--)

printf ("i=%d j=%d\n", i, j);

return 0;

}

i=0 , j=10

0 <= 10

0, 10

i=1 , j=9

1 <= 9

1, 9

i=2 , j=8

2 <= 8

2, 8

i=3 , j=7

3 <= 7

3, 7

i=4 , j=6

4 <= 6

4, 6

i=5 , j=5

5 <= 5

5 5

i=6 , j=4

6 <= 4 X

Condition false

[6.5] Nested and infinite loops

⇒ If within if - else, we have another if - else
then it is called nested if - else

⇒ loops with int loop.

for (i=1 ; i<=5 ; i++) {

 for (j=1 ; j<=3 ; j++) {

 printf ("%d %d", i, j);

}

Mail: gatecse@appliedroots.com

(i=0, j=10)

i++, j--

valid

Infinite loop : Common Mistakes

(1) `for (-i - ;) {`

=

}

- I have nothing to initialize,
nothing for condition and
update.

keep running for infinite time

(2) `while (1) {`

=

}

// always true so keep running
to infinite time.

(3) `do {`

=

}
`while (1)`

Ex:1

```
int i=1;  
while (i<=5);  
{}
```

body of the loop is empty
Semicolon is the mistake
here

Ex:2

```
for (i=0; i<=3; i--)  
{}
```

$i = 0, -1, -2, \dots$

so I will be keep decreasing
so loop will keep executing

Ex:3

while ($i=2$)

{

=

}

-

Phone: +91 844-844-0102

/* assignment not comparison
so this is always true
keep running

Ex:4

X

```
float f = 2.0;
while (f != 3.0)
{
    f += 0.2;
}
```

/* code is looking perfect but
in computers, float values
are stored as 2.999999 or
3.000001 form

Sometimes it is not stored exactly.
Instead of this you can write
 $f <= 3.0$

[6.6] break and continue

⇒ Prime no. : for a given no. 'n' if it is not
divisible by 2, 3, 4, 5 ... (n-1) then it
is prime no.

$n \% 2 ; n \% 3 ; \dots ; n \% (n-1)$ } $(n-2)$ modulo
operations.

⇒ $\rightarrow 2, 3, 4, 5, \dots, \text{sqrt}(n)$ $n = 19$

$\lfloor 4.99 \rfloor = \lfloor 4.12 \rfloor = \lfloor 4.36 \rfloor = 4$ $\text{sqrt}(n) = 4 \dots$
floor $\lceil \text{sqrt}(n) \rceil$

Mail: gatecse@appliedroots.com

#WAP to find whether a number is prime or not.

Phone: +91 844-844-0102

APPLIED ROOTS

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, n;
    printf("enter a number:");
    scanf("%d", &n);
    for (i=2; i<=sqrt(n); i++)
        if (n%i == 0)
            break;
        if (i > sqrt(n))
            printf("%d is prime(%d)", n);
        else
            printf("%d is not prime(%d)", n);
    return 0;
}
```

O/P:

N=19
enter a number: 19
19 is prime

$i = 2$	$i = 3$	$i = 4$
$19 \% 2$		

```
for () {
    for () {
        =break;
    }
}
```

\Rightarrow break only breaks out of the prime loop

Mail: gatcse@appliedroots.com

⇒ Use of continue

Phone: +91 844-844-0102

(61)

#WAPs to find sum and average of 5 positive no.

#include <stdio.h>

int main()

{

int i=1, n, sum=0;
float avg;

printf("enter 5 positive numbers: ");

while (i<=5)

{

printf("enter number %d : ", i);

scanf("%d", &n);

if (n<0)

{

printf("enter only positive numbers (%d)");

continue;

}

sum += n;

i++;

}

while (cond){

=

Continue;

=

}

do {

=

Continue;

=

} while (cond)

for (init; cond; update)

{

=

Continue;

=

}

Goto [6.7]

Phone: +91 844-844-0102

⇒ label → Identifier

⇒ goto jump only within the function.

```
#include <stdio.h>
```

```
int main()
```

```
{ int n;
```

```
printf("enter a number:");
```

```
scanf("%d", &n);
```

```
if (n%2 == 0)
```

```
    goto even;
```

```
else
```

```
    goto odd;
```

```
even:
```

```
    printf("Number is even\n");
```

```
    goto end;
```

```
odd:
```

```
    printf("Number is odd\n");
```

```
    goto end;
```

```
end:
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

%: enter a number: 1212
Number is even

③ If you have label:
Then just place ; & don't want to write anything
→ Null Statement

```
goto
```

```
goto label;
```

=
label:

forward jump

Please avoid
goto unless it
is mand

O/P when even goto end
is commented.

enter a number: 1212
Number is even
Number is odd

Mail: gatase@appliedroots.com

```

for ( . . . ) {
    for ( . . . ) {
        goto endloops;
    }
}
endloops:
    = =

```

[6.8] Switch-Case



```

switch (expr)      → int / char / long / short
{
    case C1:      → int / char / const
    case C2:      =
    case C3:      =
    case C4:      =
    default:      =
}

```

#WAP to perform arithmetic calculations On integers.



#include <stdio.h>

```
int main()
```

```
{
```

```
    char op;
```

```
    int a, b;
```

```
    printf("enter operator"),
```

```
    scanf("%s", &op);
```

```
    printf("enter two numbers:"),
```

```
    scanf("%d %d", &a, &b);
```

```
    switch(op)
```

```
{
```

Case '+':

```
        printf("Result = %d\n", a+b);
```

```
        break;
```

Case '-':

```
        printf("Result = %d\n", a-b);
```

```
        break;
```

Case '*':

```
        printf("Result = %d\n", a*b);
```

```
        break;
```

Case '/':

```
        printf("Result = %d\n", a/b);
```

```
        break;
```

Case '%':

```
        printf("Result = %d\n", a%b);
```

Phone: +91 844-844-0102

Valid

Invalid

int $i_1, i_2, i_3;$
char $c_1, c_2;$
float $f;$

Switch (i_2)

Switch (c_1)

Switch ($i_1 > i_2$)

Switch ($c_1 + c_2 - 3$)

Case 4:

Case 'a':

Case 2+4:

Case 'a' > 'b':

switch (f) X

Switch ($i_1 + 4.5$) X

Should be integer only

Case "abc": X

String is not allowed

Case i_1 : X Not allowed

This is Integer variable
not constant

Case $i_1 > i_2$:

X
If i_1 & i_2 are integer variables
so this is not allowed.

[6.9] Sample Problem [GATE 2016]

The following function computes x^y for positive integers x and y .

```
int exp(int x, int y)
{
    int res=1, a=x, b=y;
    while(b!=0)
    {
        if(b%2 == 0)
        {
            a=a*a;
            b=b/2;
        }
        else
        {
            res = res*a;
            b = b-1;
        }
    }
    return res;
}
```

```
else
{
    res = res*a;
    b = b-1;
}
}
return res;
```

Mail: gatessc@appliedroots.com

④ $x^y = a^b$
 $a=4, b=5$

if $x=4, y=5$

Ⓐ $x^y = a^b \Rightarrow 4^5 = 4^5$ could be correct

Ⓑ $(1*4)^5 = (1*4)^5$ "

Ⓒ $x^y = res * a^b \Rightarrow 4^5 = 1*4^5$

Ⓓ 1st iteration $res=4, a=4, b=4$

Ⓐ $x^y = a^b \Rightarrow 4^5 = 4^4 \times$

Ⓑ $(4*4)^5 = (4*4)^4 \times$

Ⓒ $x^y = res * a^b \Rightarrow 4^5 = 4*4^5$ 4 would be correct

Ⓓ $x^y = (res * a)^b \Rightarrow 4^5 = (4*4)^4 = 4^8 \times$

Thus so option (C) is correct.

③ Iteration - 3

$$\begin{aligned} a &= (4^2)^2 \\ b &= 1 \\ res &= 4 \end{aligned}$$

④ Iteration - 4

$$\begin{aligned} res &= 4 * (4^2)^2 = 4^5 \\ b &= 0 \\ a &= (4^2)^2 \times^4 \end{aligned}$$

Only 3-Multiplication

Mail: gatecse@appliedroots.com $= 4 * 4^2 * 4^2$

Consider the following pseudo code, where x and y are positive integers

```

begin
    q := 0
    r := x
    while r >= y do
        begin
            r = r - 1
            q = q + 1
        end
    end

```

The post condition that needs to be satisfied after the program terminates is

- (A) $\{r = qx + r \wedge 1 \leq r \leq y\}$ (C) $\{y = qx + r \wedge 0 \leq r \leq y\}$
- (B) $\{x = qy + r \wedge r < y\}$ (D) $\{q + 10\}$

$$r = x - qy \quad r < y$$

$$r + qy = x \quad r < y$$

$$x = r + qy$$

$$x = qy + r \quad r < y \quad \text{— option (B)}$$

Consider the following C program.

```
#include <stdio.h>
int main()
{
    int i, f, k=0;
    f = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
    k = --f;
```

```
for (i=0; i<5; i++)
{
```

```
    switch(i+k)
{
```

Case 1:

Case 2: printf("1H%d", i+k);

Case 3: printf("1H%d", i+k);

default: printf("1H%d", i+k);

}

}

between 0,

}

The no. of times printf statement is executed is _____

(A) 8

(B) 9

(C) 10

(D) 11

$$k=0$$

L to K

$$f = 1 + 0 \cdot 4 + 1$$

$$j = 6 / 4 + 2 \cdot 0 / 5 + 8 / 5$$

$$= 1 + 0 \cdot 4 + 1 = 2$$

↑
(typecasted to int)

$$f = k - 2 = 1$$

$$k = k - 1 = 0$$

for loop

$$i=0$$

$$i+k = 0 - 1 = -1$$

default

$$i+k = 0 - 1 = -1 \quad \text{---} \textcircled{1}$$

$$i+k = 1 - 1 = 0$$

$$\text{default: } i+k = 1 - 1 = 0 \quad \text{---} \textcircled{1}$$

$$i=2$$

$$i+k = 2 - 1 = 1$$

Case 1

— \textcircled{3}

$$i=3 \quad i+k = 3 - 1 = 2 \quad \text{---} \textcircled{3}$$

$$i=4, \quad i+k = 4 - 1 = 3 \quad \text{---} \textcircled{2}$$

Consider the following Pseudo code. What is the total number of multiplication to be performed

$$D=2$$

for $i=1$ to n do

 for $j=i$ to n do

 for $k=j+1$ to n do

$$D=D+3$$

- (A) Half of the product of 3-consecutive integers.
- (B) One-third of the product of 3 consecutive integers.
- (C) One-sixth of the product of 3-consecutive integers.
- (D) None of the above.

Solution: at $i=1$

$$\begin{array}{l} j=1 \rightarrow k=2, 3, \dots, n \rightarrow (n-1) \text{ times} \\ j=2 \rightarrow k=3, 4, \dots, n \rightarrow (n-2) \text{ times} \\ \vdots \\ j=n \end{array}$$

Sum of first n -integers $\Rightarrow n(n+1)/2$

$$i=2$$

$$j=2 \rightarrow k=3, \dots, n$$

$$j=3 \rightarrow k=4, 5, \dots, n$$

$$3(n-2)(n-1)/2$$

$$S = \frac{n(n-1)}{2} + \frac{(n+1)(n-2)}{2} + \frac{(n-2)(n-3)}{2} + \dots$$

$$2S = \{n^2 - n\} + \{(n-1)^2 - (n-1)\} + \{(n-2)^2 - (n-2)\} + \dots$$

$$= \{n^2 + (n-1)^2 + (n-2)^2 + \dots\} - \{n + (n-1) + (n-2) + \dots\}$$

$$= \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2}$$

$$2S = \frac{n(n+1)(2n+1) - 3n(n+1)}{6}$$

$$S = \frac{n(n-1)(n+1)}{6}$$

S is the product of 3 consecutive integers

Solved Problem [Gate 2012]

What will be o/p of the following c program

```
char puchar = 'A';
switch (puchar)
{
    Case 'A':
        printf ("choice A\n");
    Case 'B':
        printf ("choice B\n");
    Case 'C':
    Case 'D':
    Case 'E':
```

default:
 $\text{printf} (\text{"No choice"});$

(A) No choice

Phone: +91 844-844-0102

(B) choice A

(C) choice A

Choice B No choice

(D) Program gives no output as it is erroneous.

Solution:

switch (inchar)

↳

switch (A')

↳

Case 'A'

Case B

Case C :

Case D :

Case E :

default :

Choice A

Choice B

As there is no break
No choice

[GATE] Solved Problem [GATE 2004]

Consider the following C program (Euclid Algo for GCD)

main()

{

int x, y, m, n;

Scanf ("%d %d", &x, &y);
 $m = x;$
 $n = y;$

while ($m \neq n$)

{ if ($m > n$)

$m = m - n;$

else $n = n - m;$

Printf ("%d", n);

(A) $x+y$ using repeated subtr

(B) $x \bmod y$..

(C) the gcd of $x \& y$

(D) LCM of $x \& y$

Solution: Pick $x \neq y$

$$\textcircled{A} \quad x+y = 2$$

$$\textcircled{B} \quad x \bmod y = 0$$

$$\textcircled{C} \quad \text{GCD}(x,y) = 3$$

$$\textcircled{D} \quad \text{LCM}(x,y) = 6$$

If $x=18, y=12$

$$\textcircled{A} \quad 18 \div 12 = 1$$

$$\textcircled{B} \quad 18 \bmod 12 = 6$$

$$\textcircled{C} \quad \text{GCD}(18,12) = 6$$

$$\textcircled{D} \quad \text{LCM}(18,12) = 36$$

$$x=6, y=3$$

$\frac{4}{3}$

$$m=6, n=3$$

$$6! = 3 \quad \checkmark$$

$$\text{if } (6 > 3) \quad \checkmark$$

$$m=6-3=3$$

$$n=3$$

$$3! = 3 \quad \times$$

$$\text{Print}(n)$$

$$\frac{4}{3}$$

So ans is 3

[6.15] Solved Problem [Gate 2014]

Consider the function func shown below:

```
int func(int num)
{
    int count = 0;
    while (num)
    {
        count++;
        num >>= 1;
    }
    return (count);
}
```

- A 8
- B 9
- C 10
- D 11

Mail: gatecse@appliedroots.com
The value returned by func(435) is —

func(435)
 ↓
 count = 0

Phone: +91 844-844-0102
 $\begin{array}{ccccccccc} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}$
 $(\overline{110110011})_2$

while(435)
 ↓
 count++ ;
 435 >>= 1 ;

Shifting 1-digit right

$\begin{array}{ccccccccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$

No. of times we need to shift so number will become zero.

F. Functions

⇒ functions in C

[F.1] Introduction to Functions

isPrime(int num)

Why do we need functions? Functions basically allow us to break up & organise code to logical & modular format.

printf, scanf, sqrt → libraries & header file

Adv : to reuse

{ main()
 === }

There are broadly two types of functions: **library functions** → user-defined function

↳

`printf, scanf`

`Sqrt`

↳

`#include <stdio.h>`

Preprocessor directive

`#include <math.h>`

header file & library file
declare definition

WAP to find the Sqrt of a number.

`#include <stdio.h>`

`#include <math.h>`

`int main ()`

{

`double n, s;`

`printf("enter a num: ");`

`scanf("%f", &n);`

`s = Sqrt(n);`

`printf("Sqrt of %.2lf is %.2lf\n", n, s),`

`between 0,`

}

`double Sqrt (double)`

↳

`int isPrime(int n)`

{

=

{

}

return 1 if n
is prime and
0 if n is not
prime.

O/P: enter a num: 17

Sqrt of 17.00 is 4.12

#WAP to find the sum of two numbers.

Phone: +91 844-844-0102

#include <stdio.h>

int main() → int sum(int x, int y); // fn declaration

{ int a, b, s;

printf ("enter values for a and b:"); Calling for
scanf ("%d %d", &a &b);

s = sum(a, b);

printf ("sum of %d and %d is %d\n", a, b, s); // Calling function sum

return 0;

int sum(int x, int y) // function definition

{ int s;
s = x + y;
return s;

main → a, b, s
called fn
↓
45 34

Sum(45, 34)
x y

$$s = 45 + 34 = 79$$

Syntax:

{ between type fn ^{identifier} name (type1 arg1, type2 arg2)

local variables declaration ;

body

return (exp); // return is also optional
can be enclosed w/o parenthesis

void - not going to return anything
int - default (optional)

Mail: gatecse@appliedroots.com

⇒ If I define calling func before called func
then I need to declare called function.

⇒ If I define called func above the calling func
then I need not to declare called function.

```
#include <stdio.h>
int sum (int x, int y)
{
}
int main()
{
    = sum();
}
```

called funcⁿ

calling funcⁿ

⇒ function declaration is optional not mandatory.

⇒ As soon as the func returns to the calling function, all the variables (arguments, local variable) will be removed from the memory and you can not access it.

① We can not define one function within another function.

```
int main()
{
    int sum()
}
```

X Not allowed

② You can have different **functions** defined & declared in various files.

③ Only one variable can be returned.

return a, b; X - invalid

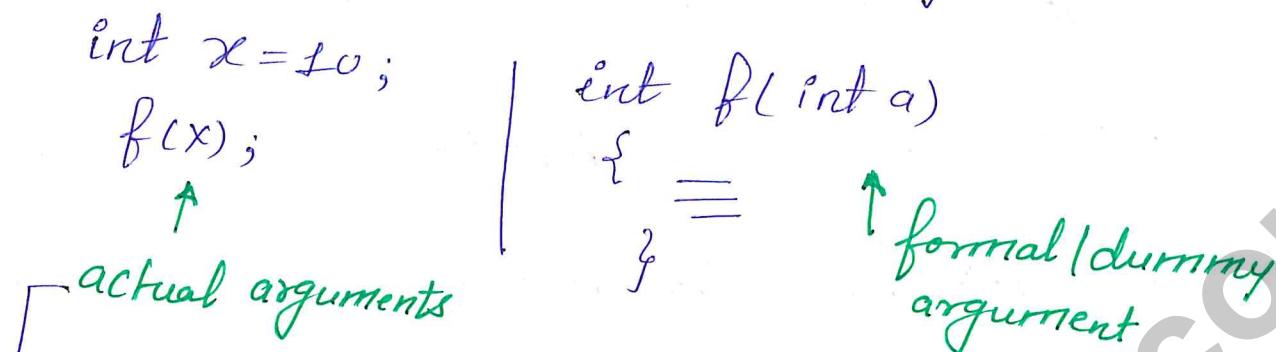
WAP to understand the use of return statement.

```
#include <stdio.h>
void selection (int age, float ht); // fn d
int main ()
{
    int age;
    float ht;
    printf ("enter age and height:");
    scanf ("%d %f", &age, &ht);
    Selection (age, ht);
    return 0;
}
void selection (int age, float ht)
{
    if (age > 20)
    {
        printf ("Age should be less than 20 in");
        return;
    }
    if (ht < 5)
    {
        printf ("height should be more than 5 in");
        return;
    }
    printf ("Selected in");
}
```

[7.2] Let's dive into functions | **Phone: +91 844-844-0102**

**APPLIED
ROOTS**

⇒ actual arguments vs formal arguments



variable,
const, expr,
fn-call

$f(x); \quad f(2); \quad f(2+3+x);$

⇒ There are four types of functions?

- | | | |
|-----------|-----------|--------|
| ① no args | no return | } void |
| ② args | no return | |
| ③ no args | returns | |
| ④ args | returns | |

Rule

① Can declare a function within another function

② Declaration → if a funcn returns int/char
we need not declare pt.

Mail: compiler@appliedroots.com

③ Always declare function **Phone: 191844-844-0102**

④ no-args → int f() ← declare
int f(void) ← explicit way of telling

HO- return \rightarrow void f(void);

⑤ Actual arguments & formal arguments

⑥ num of actual args > num of formal args

int f(int a); → $f[2, 3]$
+ ignored

⑥ # formal args > # actual args

int f (int a, int b); | f (2);
 ↑
 garbage

⑥ What if there is data type mismatch

```
double sqrt (double d); | sqrt (3);  
                      ↑ |  
                      |  
                      int
```

C Compiler will try to force type cast
(It will try and do implicit type-casting)
int → double ✓

If type - Casting doesn't work then it will give some garbage value.

`Print f(int a);`

`f(2+23)`

`double/float`

try to type-cast

`int f(float x)`

`f("abc")`

no way to type-cast a string into float

`Sqrt((double) 3.)`

[6] `int m=3, k;`

`k = add(m, m++)`

`3, 3 → 6`

`4, 3 → 7`

{ order of evaluation of argument }

Not defined in C

Compiler dependent

(Please avoid code like this)

[7] `main()`
return → int
 $\hookrightarrow 0 \Rightarrow$ terminated successfully
 Non-Zero ⇒ errors

\Rightarrow OS calls the main function & returns goes back to declaration → C-compiler

definition → programmer / user

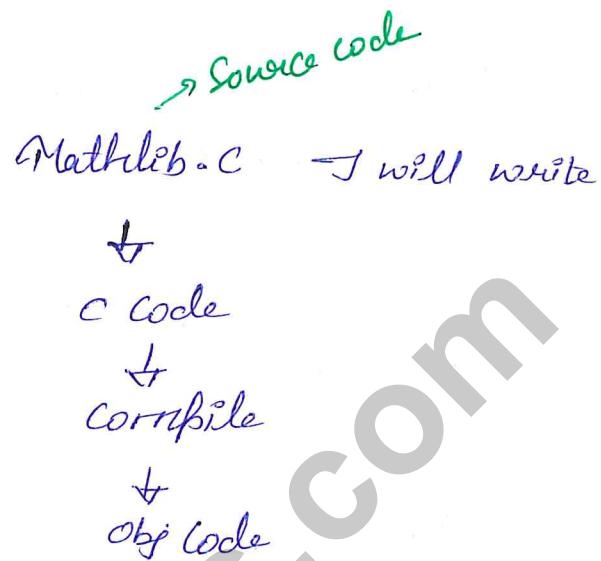
between 0;

`exit(0);` args (command line args)

⑧ Library function: `Printf`, `scanf`: [Phone: 844-844-0102](tel:844-844-0102)



Call - Programmer
declared → header-files
definition → Libraries → obj code



⑨ Older versions → Supported by modern compiler

Declaration:

ret-type fn-name();
└ Argument list empty

⇒ Compiler needs return types & function name mostly.

```
ret-type fn-name(a,b,f)
{
    int a,b;
    float f;
}
```

gcc

7.3] Local, Global and Static Variables

Phone: +91 844-844-0102



Global variables are accessible everywhere

```
#include <stdio.h>
```

```
void func1(void)
```

```
void func2(void)
```

```
int a, b = 6; // outside, global-var b=6
```

```
int main()
```

```
{
```

```
    printf("Inside main(): a=%d, b=%d\n", a, b);
```

```
    func1(); // control goes to func1() body 0 6
```

```
    func2(); // after ending func1() when it come back then
```

```
    return 0; // func2() is called.
```

```
}
```

```
void func1(void)
```

```
{
```

```
    printf("Inside func1(): a=%d, b=%d\n", a, b);
```

```
void func2(void)
```

```
{
```

```
    int a = 8; // local variable declared & initialized
```

```
    printf("Inside func2(): a=%d, b=%d\n", a, b);
```

```
}
```

O/P:

```
Inside main(): a=0, b=6
```

```
Inside func1(): a=0, b=6
```

```
Inside func2(): a=8, b=6
```

#WAP to understand the use static variable
Phone: +91 844-844-0102

```
#include <stdio.h>
void func();
int main()
{
    func();
    func();
    func();
    return 0;
}
void func(void)
{
    int a=10; // local variable
    Static int b=10; // static local variable
    printf("a=%d, b=%d\n", a, b);
    a++;
    b++;
}
```

main()



func()



a=10

Static b=10

Pf = a=10, b=10
a++; - 11
b++; = 11

func()



a=10

Static b=11

| Pf a=10, b=11
a++; 11
b++; 12

func()



a=10

Static b=12

Pf a=10, b=12
a++ = 11
b++ = 13

[7.4] Recursion : the basics

Phone: +91 844-844-0102



Real World

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1 \quad (\text{repetition})$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

+
loops

+
Iterative solution

int factorial (int n);

int i, ret = 1;

{
for (i=1; i<=n; i++) → if (n==0) return 1;
}

ret = ret + i;

}

return ret;

}

Recursive : simple way of solving large problem.

$$n! = n * (n-1)!$$

$$(n-1) * (n-2) * (n-3) \dots 1$$

factorial (n) = $n * \text{factorial}(n-1)$ if $n \neq 0$ → recursive case

| if $n = 0$ → base case

or
termination case

$$\text{factorial}(3) = 3 * \text{factorial}(2)$$

$$3 * \text{factorial}(1)$$

Mail: gatecse@appliedroots.com

$$1 * \text{factorial}(0) \rightarrow 1$$

int factorial (int n)

Phone: +91 844-844-0102

{
if ($n == 0$)

return 0;

else

return ($n * factorial(n - 1)$);

} Winding &
unwinding }

main()

{

int x = factorial(3);

6

factorial(3)

↓

$n = 3$

return ($3 * factorial(2)$);

3*2

factorial(2)

↓
 $n = 2$

return ($2 * factorial(1)$);

1

factorial(1)

↓
 $n = 1$

return ($1 * factorial(0)$);

↓
 $n = 0$

↓
 $n = 1$;

Fibonacci numbers :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

$fib(n) = fib(n-1) + fib(n-2)$ if $n \geq 2$
recursive case

0 if $n == 0$

1 if $n == 1$

base / termination case

Mail: gatecse@appliedroots.com

int fib(int n)

```

    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return (fib(n-1)+fib(n-2));
    }
}
```

Phone: +91 844-844-0102

→ Code is trivial if we have the recursive definition

- Simplifying the code

① recursion → Iteration

② recursion → More function calls
↳ less-efficiency

{ DS & algo }

↳ how recursion works internally → stack

③ tail recursion
→ objective of tail recursion is to make recursion more efficient

+
call stack

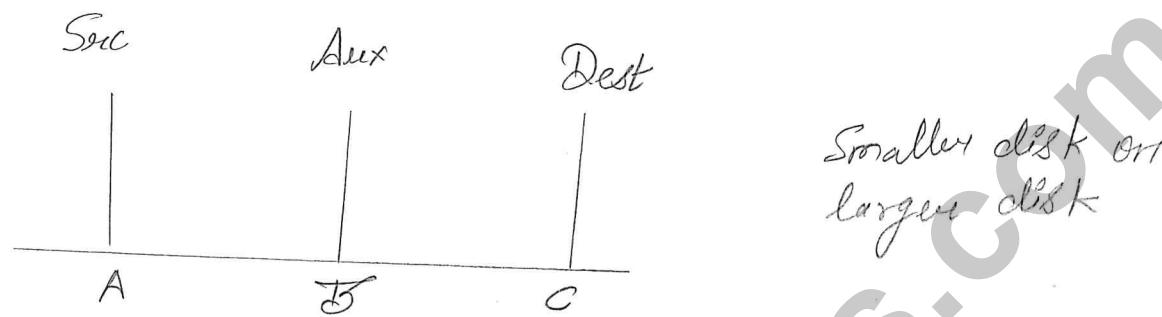
[7.5] Recursion: Tower of Hanoi

81

Phone: +91 9444-844-0102



- ⇒ Tower of Hanoi is the mathematical game or puzzle
- ⇒ Good example of recursion.



- ⇒ Indian temple Kashi Vishwanath, surrounded by 64 golden disks.
- $$2^{64} - 1 = 585$$
- ⇒ Monastery in Hanoi, Vietnam.

for 1 disk : for moving from one tower to other.
 $n=1$

for 2-disks : for moving from one src tower to dest tower
 $n=2$

- Move smaller disk to Aux
- Move larger disk to Dest.
- Move smaller disk from Aux to dest

Breaking a larger problem into smaller size

Recursion → Pattern → recursion equation → code

Mail: gatecse@appliedroots.com

3-disk : 7 Moves

Phone: +91 844-844-0102



$(n, \text{Src}, \text{dest}, \text{aux})$



$(n-1), A, B, C \rightarrow 1$

+ $n^{\text{th}} \text{ disk} : A \rightarrow C \rightarrow 2$

+ $(n-1) \text{ disk} : B, C, A \rightarrow 3$

↳ Multiple

$$\text{tob}(n, \text{Src}, \text{dest}, \text{aux}) = \begin{cases} n^{\text{th}} \text{ disk} \\ \text{Src} \rightarrow \text{dest} & , n=1 \\ \text{tob}(n-1, \text{Src}, \text{aux}, \text{dest}) \\ + n^{\text{th}} \text{ disk} : \text{Src} \rightarrow \text{dest} \\ + \text{tob}(n-1, \text{aux}, \text{dest}, \text{Src}) \end{cases}$$

↓
recursive case

#include <stdio.h> Phone: +91 844-844-0102

void toh (int n, char src, char dest, char aux)

{ if (n==1){

 printf ("In move disk 1 from %c to %c", src, dest);

 return;

}

 // base case

 toh (n-1, src, aux, dest);

 printf ("In move disk %d from %c to %c", n, src, dest);

 toh (n-1, aux, dest, src);

}

int main ()

{ int n=4;

 toh (n, 'A', 'C', 'B');

 return 0;

}

[7.6] Storage Classes : auto, extern, static, register

#include <stdio.h>

void func ()

{ int x=2, y=5; // auto int x=2, y=5;

 keyword

 printf ("x=%d, y=%d\n", x, y);

 x++; y++;

int main ()

{

 func();

 func();

 func();

O/P: x=2, y=5
 x=2, y=5
 x=2, y=5

When I come out of func(), memory is deleted

⇒ Any variable which is declared ~~inside~~ outside ~~9101-844-0102~~ auto by default.

⇒ Lifetime of a variable is time between creation and destruction of a variable.

⇒ Scope of a variable is where can a variable be accessed.

Lifetime: block in which they are present and after they are declared

Scope: Can only access variables after they are declared & can be used with in this block only.

Init: Initialize to garbage if you don't explicitly initialized.

Default auto int a;

#include <stdio.h>

int main (void) {

int i=10;

{ int i=5;

printf("In inside block: %d", i);

printf("In outside block: %d", i); O/p

Mail: gatcse@appliedroots.com

inside block 5
outside block: 10

Extern: external storage classes
keyword

⇒ If you have variables that are used by many functions or across multiple files

```
int x=8; //definition
main() {
    =
}
f1() {
    =
}
f2() {
    =
}
```

main()

extern int x; // Then x will be
accessible in main

f1()

extern int x;

int x=8; // defi

f2()

{

}

Not accessible
here

global var

accessible here

⇒ extern int x; says this variable is present and it is created elsewhere.

file1.c

```
int x=8; //definition
main() {
```

}

f1()

file2.c

extern int x; //declaration

f2()

{

}

f3()

a variable type int
is created elsewhere

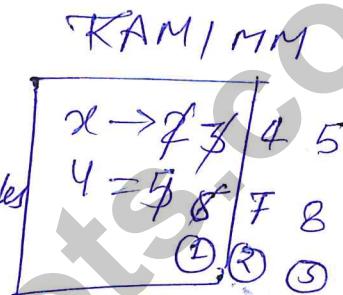
① `extern` keyword is helping to increase the scope or region of the code where I can access this variable.

② `extern` → init to Zero (by default).

Static : initialized to zero (by default)

```
void main()
{
    static int x=2, y=5; // local static variables
    printf("x=%d, y=%d\n", x, y);
    x++; y++;
}
```

```
int main()
{
    func();
    func(); can not use x here.
    func(); (out of scope)
    return 0;
}
```



func()
 $x=2, y=5$
 pf(2, 5)
 $2++;$ 3
 $5++;$ 6 } ①

func() func() func()
 $x=3$ $y=6$ $x=4, y=7$
 pf(3, 6) pf(4, 7)
 $3++$ 4 } ② $4++$ 5 } ③
 $6++$ 7 } ④ $7++$ 8 } ⑤

As at the end of function x & y would have been removed from memory but here we are using static so they will not be removed from memory.

Lifetime of a static variable : from the time function first called till the time program ends.

Scope of static variables : Region of code in which I can access the

⇒ I can initialize static **Variables** as **Constants** or as Constant expressions.

Static int $x=2;$
 $(2+3);$

Static int $x=a;$ X Can not initialize static variable using another variable

⇒ And also initialization of static variable is happened only when first time they called. And second time if we call they are already present in m/r they are not initialized and no memory is created.

Global Static variables, which are called out of all functions

```
#include <stdio.h>
Static int y=5;
Void func()
{
    int x=2;
    printf("x=%d, y=%d\n", x, y);
    x++; y++;
}
Int main()
{
    func();
    func();
    func();
}
```

$$\begin{array}{ll} \text{O/P:} & x=2, y=5 \\ & x=2, y=6 \\ & x=2, y=7 \end{array}$$

file1.c

```
int x=8; //global non-static
```

```
Static int y=10;
```

main {

f1() {

file2.c

```
extern int x;
```

f2() {

f3() {

}

Not allowed y in f2() & f3()

⇒ scope of static variables is limited within the file.

⇒ Lifetime: throughout the program.

Register

: keyword in C.

Cache / register processor

```
#include <stdio.h>
int main()
```

{

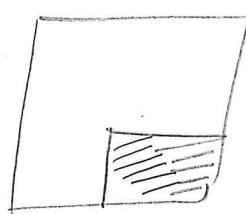
register int i;

for (i=0; i<100; i++)

printf("%d\n", i);

return 0;

Processor



RAM

64B

↓ MR (Cache) [MB]

↓ Registers (Older terminology)
for this memory

Storage classes functions Phone +91 844-0102 [default storage class for]

Static

file1.c

main() { }

f1() { }

file2.c

extern void f1(); // Not mandate
f2() { }

Static f3() { }

↓
Static global variable

[T.5] Solved Problems: 1 to 5 [GATE 2019, 2019, 2016, 2015, 2016]

Consider the following C program:

```
#include <stdio.h>
int jumble(int x, int y) {
    x = 2 * x + y;
    return x;
}

int main() {
    int x = 2, y = 5;
    y = jumble(y, x); -12
    x = jumble(x, y); -26
    printf("%d\n", x);
    return 0;
}
```

Start with main()

$$x = 2, y = 5$$

$$y = jumble(5, 2)$$

$$\begin{matrix} & 1 \\ x & \downarrow \\ & 4 \end{matrix}$$

$$x = 2 * 5 + 2 \\ = 10 + 2 = 12$$

$$x = jumble(12, 2)$$

$$\begin{matrix} & 1 \\ x & \downarrow \\ & 4 \end{matrix}$$

$$x = 2 * 12 + 2 = 26$$

The value printed by the program is 26

Consider the following C function. **Phone: +91 844-844-0102**



`void convert (int n) {`

`if (n < 0)`

`printf ("%d", n);`

`else {`

`convert (n/2);`

`printf ("%d", n % 2);`

`}`

`}`

`c(40)`

$n < 0 \times$

`c(5)`

\downarrow

$n < 0 \times$

`c(2)`

\downarrow

$n < 0$

`c(1)`

\downarrow

$n < 0 \times$

`c(0)`

\downarrow

`c(0)`

Consider the following program **Program No 1 844-844-0102**

```
#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);
```

Starting from main()

↓

calc (4, 81);

↓

a = 4, b = 81

Counter++; 1

if (81 == 3) X

c = calc (a, 27)

↓

a = 4, b = 27

Counter++; 2

if (27 == 3) X

c = calc (3, 9)

↓

a = 3, b = 9

Counter++; 3

if (9 == 3) X

c = calc (4, 3)

Counter++; 4

if (3 == 3) V

return (4 * 4 * 4) = 64

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 > j / 2; i = i / 2) j++;
```

```
    for (j >= 1; j = j / 2) sum++;
```

```
    return sum;
```

Mail: gatecse@appliedroots.com

The value returned when n will have with $i^{\frac{1}{2}}$ & $j^{\frac{1}{2}}$ in ?.

fun(2⁴⁰)

Phone: +91 844-844-0102



, j=0, sum=0

for (i=2⁴⁰; i>1;

for (j=j>1; j=j/2) sum++

j=40 → sum=1

j=20 → sum=2

j=10 → sum=3

j=5 → sum=4

j=2 → sum = 5

j=1 → sum=6

j=j++ = 1

i=2⁴⁰ → j=1

i=2³⁹ → j=2

i=2³⁸ → j=3

!

i=1 → j=40

i=0 → j=41

i>1 loop termin

Consider the following program written in Pseudo code.
Assume that x & y are integers.

```

Count(x,y) {
    if (y!=1) {
        if (x!=1) {
            printf(" * ");
            Count(x/2,y);
        }
        else {
            y=y-1;
            Count(1024,y);
        }
    }
}
```

C(1024,1024)

(1024 != 1) x

(1024 != 1) x

printf(" * ") — ①

C(512,1024)

* — ②

C(256,1024)

* — ③

C(128,1024)

, * — ④

!

C(1,1024) — 10 times

Now else part

The no. of times that the printf statement is executed by the call Count(1024,1024) is _____

If ($y! = 1$) $\leftarrow C(1, 1_{1024})$

If ($x! = 1$) \times

else

$$y = y - 1 = 1_{1023}$$



$C(1_{1024}, 1_{1023})$



$C(1, 1_{1023})$

Phone: +91 844-844-0102

$C(1, 1_{1023})$

$$\downarrow \quad y = 1_{1022}$$

$C(1_{1024}, 1_{1022})$



$C(1, 1_{1022})$



$\Rightarrow 10 \rightarrow 1_{1024}$

$10 \rightarrow 1_{1023}$

$10 \rightarrow 1_{1022}$

|
|
|
 $10 \rightarrow \underline{\$}$



$$1023 \times 10 = 10230$$

Chapter \Rightarrow 9 Pointers

\Rightarrow one of the most powerful concept in C

[9.1] Introduction to Pointers

Pointers make programmers super powerful
how the MM or RAM looks

$4GB \rightarrow 4 \times 2^{32}$ Bytes

address - 0 to $4 \times 2^{30} - 1$

$$1KB = 2^{10} B = 1000$$

$$1MB = 2^{20} B = 1 \text{ Million}$$

$$1GB = 2^{30} B = 1 \text{ Billion}$$

$$1TB = 2^{40} B = 1 \text{ Trillion}$$

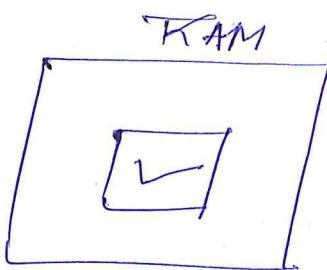
Mail: gatcse@appliedroots.com

01

When you give me address, I can directly access it in RAM

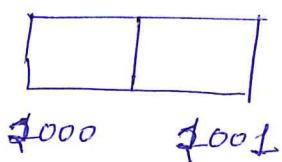
Phone: +91 844-844-0102

⇒ sequential access



int i; // variable declaration

2B Compiler allocated some memory in RAM



i = 10;



Address operators: There are two operators:

Ampersand & :
*

Ampersand (&): Unary operator

Pointf("%d", &i);
%P[↑]_(pointer) → address of variable i

⇒ addresses are whole numbers.

⇒ Pointer stores address of variable.

2i; address of variable ✓

2123; X

Correct
Mail: gatecse@appliedroots.com

Pointer Variable:

Phone: +91 844-844-0102

int * Pint;

float * Pf;

→ pointer variable of integer type

pointer variable called Pf which can point to variables float.

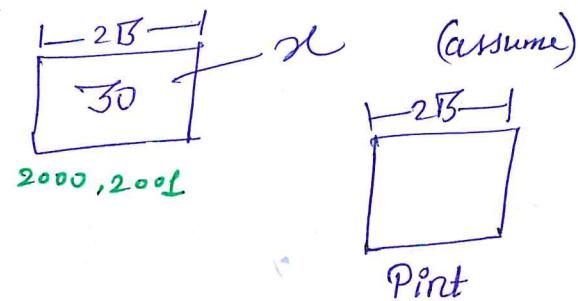
⇒ Pint, Pf are also variables, so there is some memory is allocated to store these variable also. Generally it is 2B, but it is computer specific.

⇒ int i, f, * Pint;

Assignment of pointer variables

int * Pint, x=30;

Pint = &x; // assignment
2000

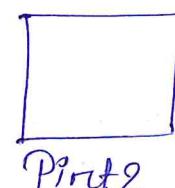


float f=1.0;

Pint = &f; → Not compile error
→ Wrong output

② int * Pint2 = Pint;

I can assign pointer to another pointer.



③ Pint = NULL → zero = 0

↑ const present in stdio.h

Mail: gatecse@appliedroots.com

0 is boundary case which says not point to anything.

03

[9.2] Dereferencing of pointers +91 844-844-0102



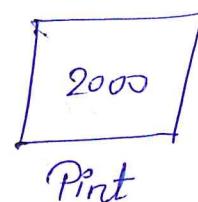
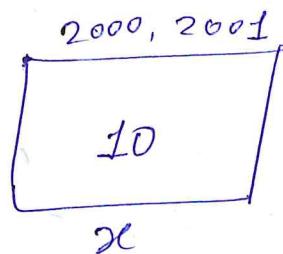
* : Indirection operator (value stored at) ; Unary

int $x = 10;$

int *Pint = &x;

*Pint = 9;

Value stored
at this address
location



value stored at 2000 = 9
 $x = 9$

② $(*Pint)++; \Rightarrow x++;$

③ printf("%d", *Pint); $\rightarrow 10$

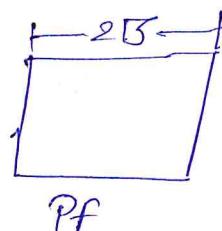
④ scanf("%d", Pint);

\downarrow
 $\&x$

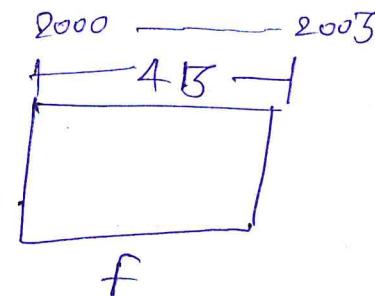
⑤ $*(\&x) = x$
*(2000) value that is stored here

Syntax : datatype *Pname

float *Pfnt;



float f = 1.0;



Pf = &f;

printf("%f", *Pf);

Indirection operation

Mail: gatcse@appliedroots.com

`int * Pint, i=10;`

`Pint = &i;`

Phone: Itself **3412410102** contains 2000

`Sizeof(*Pint); // sizeof (int)`

`Sizeof(Pint); // sizeof`

$\frac{1}{4}$
4B

`#include <stdio.h>`

`int main ()`

`{`

`int a=87;`

`float b=4.5;`

`int * P1=&a;`

`float * P2=&b;`

`printf ("value of P1 = Address of a = %p\n", P1);`

`printf ("value of P2 = Address of b = %p\n", P2);`

`printf ("value of P1 = %p\n", P1);`

`printf ("address of P2 = %p\n", P2);`

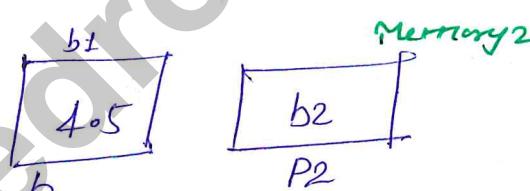
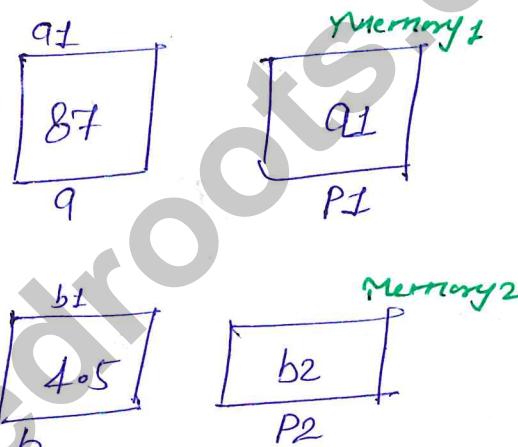
`printf ("value of a = %d %d %d\n", a, *P1, *(P1));`

`printf ("value of b = %f %f %f\n", b, *P2, *(P2));`

`return 0;`

`}`

`%P` → using this, we get hexadecimal value



[9.3] Let's dive into pointers **Phone: +91 844-844-0102**



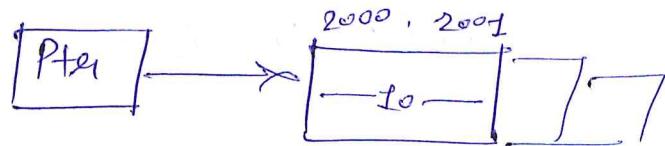
Pointer Arithmetic

(Memory is contiguous)

`int x = 10;`

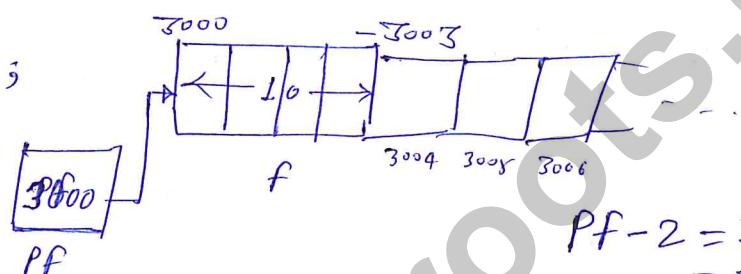
`int *Pte = &x;`

$Pte + 1 \Rightarrow 2000 + 1 * \text{sizeof}(int) = 2002$



`float f = 1.0;`

`float *Pf = &f;`



$Pf + 1 \Rightarrow 3000 + 1 * (\text{sizeof}(float)) = 3000 + 4 = 3004$

$$\begin{aligned} Pf + 1 &= 3000 - 2(4) \\ &= 3000 - 8 \\ &= 2992 \end{aligned}$$

valid

`P + 2;` ✓

`P - 1;` ✓

`P++;` ✓

`P--;` ✓

`P1 - P2;` ✓

`Pte - 2;`

$= 2000 - 2 * (\text{sizeof}(int))$

$= 2000 - 2 * (2)$

$= 2000 - 4$

Mail: gatecse@appliedroots.com

Not valid

`P1 + P2;` ✗

`P1 * P2;` ✗

`P1 / P2;` ✗

`P * 2;` ✗

`P / 2;`

`P + float or double` } ✗

`P *` ..

`P /` ..

Pointer Arithmetic

Phone: +91 844-844-0102

#include <stdio.h>

int main()

{ int a = 5, *Pi = &a;

char b = 'x', *Pc = &b;

float c = 5.5, *Pf = &c;

printf ("value of Pi = Address of a = %p", Pi);

printf ("value of Pc = Address of b = %p", Pc);

printf ("value of Pf = Address of c = %p", Pf);

Pi++;

Pc++;

Pf++;

printf ("Now value of Pi = %p", Pi);

printf ("Now value of Pc = %p", Pc);

printf ("Now value of Pf = %p", Pf);

Precedence

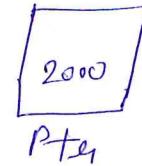
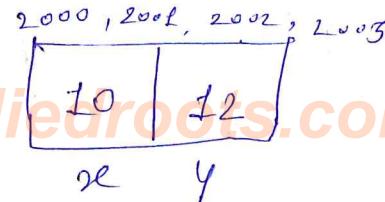
$\text{++}, \text{--}, *, \&$ (Indirection), $\&$ (address)
 Unary \rightarrow Same precedence
 (R to L) associativity

int x, y;

x = 10;

y = 12;

int *Pte = &x;

Mail: gatcse@appliedroots.com

07

$$x = * \text{ptr} + +$$

\downarrow same
 $*(\text{ptr} + +)$

Right Rhind Left 91 844-844-0102

$$x = * \text{ptr} = 10$$

$$\text{ptr} = \text{ptr} + 1 = 2002$$

(2) $x = * + + \text{ptr}; \Rightarrow * (+ + \text{ptr})$

$$\text{ptr} = 2002$$

$$x = * (2002) = 12$$

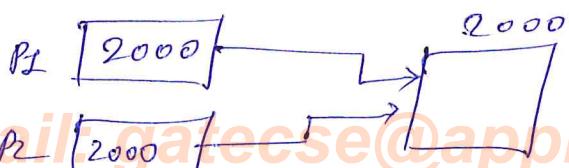
(3) $x = (* \text{ptr}) + +$ // value at this address will get increment
 $x = 10$
 $\text{ptr} = 2000$
 $\therefore x = 11$

(4) $x = + + * \text{ptr};$
 $= + + (* \text{ptr});$
 $x = 11$ and ptr will remain 2000

Comparison between pointers

\equiv \neq , $<$, $<=$, $>$, $>=$

NULL or same address or same type
 It will return true



Mail: gatcse@appliedroots.com

void pointer is basically a pointer of void type - can compare void pointer with any type of pointer

$\Rightarrow <, <=, >, >=$ operators are valid only if both the pointers are of same type or null type.

True $P_1 == P_2$

int *P1 = NULL;
float *f1 = NULL;

$P_1 == f_1$ true because both have value NULL.

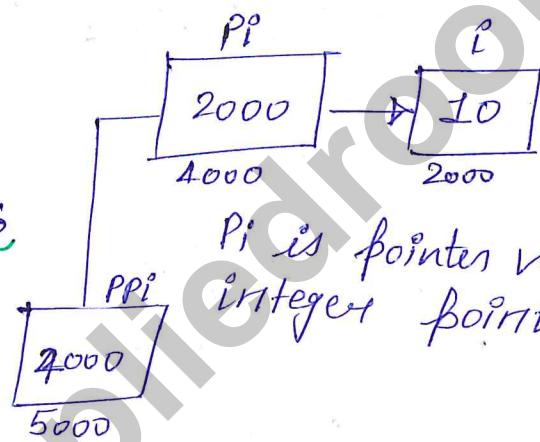
Pointers to pointers

int i = 10;

int *Pi = &i;

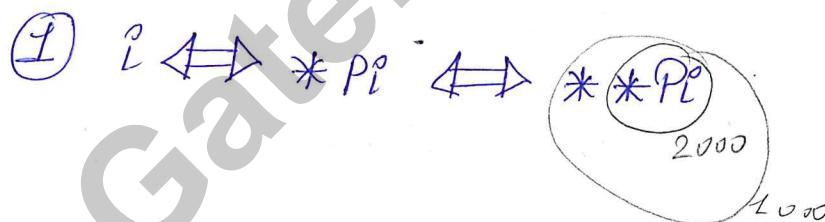
int **P2 = Π

Pointer to a pointer



Pi is pointer variable of type integer pointer.

Syntax: datatype ***Var-Name ;



② $\&i \leftrightarrow Pi \leftrightarrow *P2 \rightarrow 2000$

[9.4] Pointers and Array

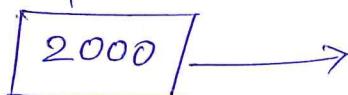
Phone: +91 844-844-0102



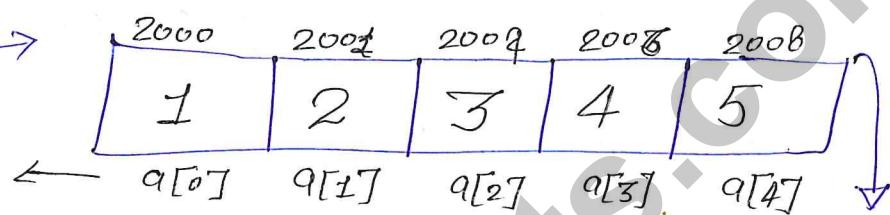
char str[10]; basic examples → Arrays

int a[5]; // Constant pointer of type int
// array declaration

datatype name size
a



Zero indexed arrays



int a[5] = {1, 2, 3, 4, 5}; // Initialize

configurations

printf("%d", a[0]);

⇒ a is the constant pointer of type int, which is pointing to very first element by storing the address of a[0] itself.

ith → a[i-1]

$$\&a[0] = 2000$$

$$\&a[1] = 2002$$

WAP to print the value and address of the elements of an array:

```
#include <stdio.h>
int main()
{
    int arr[5] = {5, 10, 15, 20, 25};
    int i;
    for(i=0; i<5; i++)
    {
```

```
        printf("value of arr[%d] = %d\n", i, arr[i]);
        printf("address of arr[%d] = %p\n", i, &arr[i]);
    }
```

Mail: gatcoos@appliedroots.com

... Printf ("address of arr[%d] = %p\n", i, &arr[i]);

value of $a[0] = 5$
 value of $a[1] = 10$
 value of $a[2] = 15$
 value of $a[3] = 20$
 value of $a[4] = 25$

address of $a[0] = 0x __ _ 0$
 address of $a[1] = 0x __ _ 4$
 address of $a[2] = 0x __ _ 8$
 address of $a[3] = 0x __ _ c$
 address of $a[4] = 0x __ _ e$

$a \rightarrow$ const pointer to 0th element. $\&a[0] \rightarrow 2000$
 $a+1 \rightarrow$ const pointer to 1st element. $\&a[1] \rightarrow 2002$
 \vdots
 $a+i \rightarrow \text{ " } \quad \text{to } i\text{th element. } \&a[i] \rightarrow$

$$*a \rightarrow 1 = a[0]$$

$$*(a+1) \rightarrow 2 = a[1]$$

$*(a+i) \rightarrow a[i] \quad // \text{this is core relationship between arrays \& pointers in C}$

whatever is written in the form of array, has an equivalent representation in terms of pointers.

$$\begin{array}{c} a[i] \\ | \\ *(a+i) \end{array} \Leftrightarrow \begin{array}{c} *(a+i) \\ | \\ *(a+i) \end{array} = \begin{array}{c} *(i+a) \\ | \\ i[a] \end{array}$$

int main ()
 APPLIED
 ROOTS

```

  int arr[5] = {5, 10, 15, 20, 25};
  int i = 0;
  for (i = 0; i < 5; i++)
  {
    printf ("value of arr[%d] = ", i);
    printf ("%d\n", arr[i]);
    printf ("%d\n", *(arr + i));
    printf ("%d\n", *(i + arr));
    printf ("%d\n", i[arr]);
    printf ("Address of arr[%d] = %p\n", i, &arr[i]);
  }
  return 0;
}
  
```

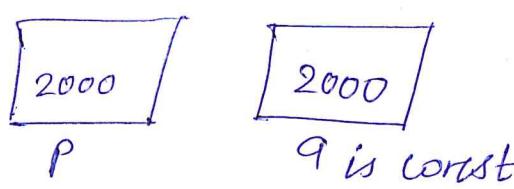
O/P:

value of arr[0] = 5 5 5 5

Address of a[0] = 0x 0

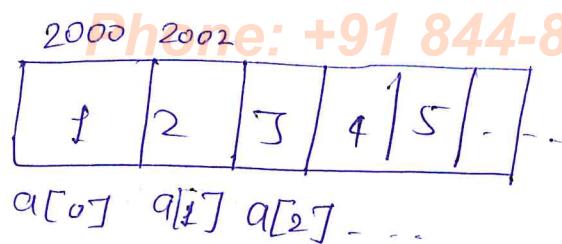
value

`int *p = a;`



`a` → array name → const
`P` → `Ptr` →

`P = #` ✓
`P++;` ✓
`P = P - 1;` ✓



$$\ast(P) = 1$$

$$\ast(P+1) = 2$$

$$2000 \leftarrow P = P+1$$

$$2 \leftarrow \ast P$$

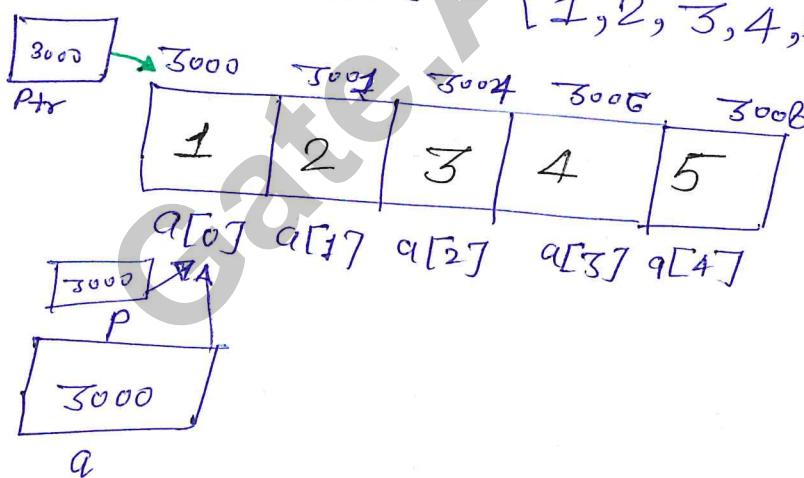
`a = #` ✗
`a++;` ✗
`a = a - 1;` ✗

Pointers to an Array

`int *p;` // integer pointer

`int (*Ptr)[5];` // pointer to an int array of size 5.

`int a[5] = {1, 2, 3, 4, 5};`



Data type `(*varname)[size];`

`P = a;` 0th element

`Ptr = a;` whole array

`P++;` → $P = 3002$

`Ptr++;` → $Ptr = Ptr + 1$

$$\Rightarrow Ptr + 1 * \text{sizeof}(5 \times 2)$$

$$\Rightarrow 3000 + 10 \\ = 3010$$

13

[9.5] Pointers and 2D PhArray +91 844-844-0102

Declaration of 2D array:

Int a[3][4]; // 2D array

1D array as a sequence
of numbers.

Initialization:

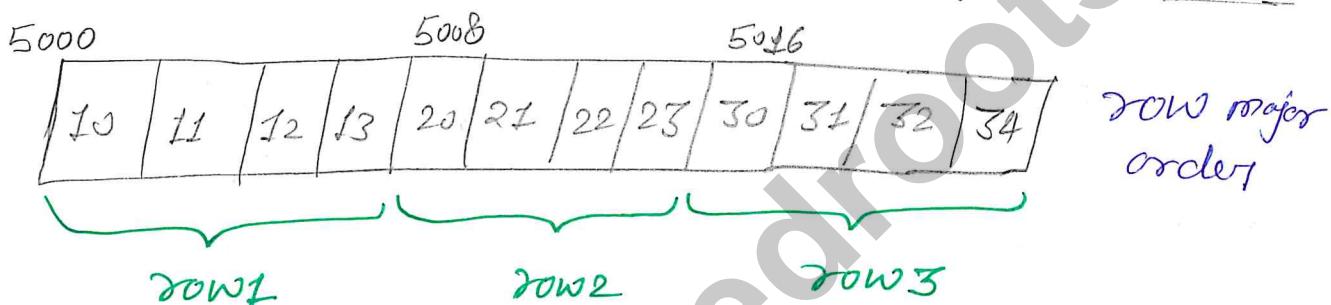
Int a[3][4] = { { 10, 11, 12, 13 },
 { 20, 21, 22, 23 },
 { 30, 31, 32, 34 } };

3 - 1D arrays &
each array contains
4 - elements.

	Col 1	Col 2	Col 3	Col 4
Row 1	10	11	12	13
Row 2	20	21	22	23
Row 3	30	31	32	34

How it is stored in Memory?

Logical representation



a[0][3]; // goto 0th row & 3rd column

a[0][3] = 13 (in array)

a[0][3] = 12

⇒ C uses row-major order. want to change as 12

Int a[3][4];

5000
a is the pointer to
the 0th row - 1D array

5008

(a+1) : pointer to 1st row - 1D array
 5016
 (a+2) : pointer to 2nd row - 1D array

	Col 0	Col 1	Col 2	Col 3
Row 1	10	11	12	13
Row 2	20	21	22	23
Row 3	30	31	32	34

{ size of each row = 4 }

$a[0] \rightarrow *(\alpha + 0) = 5000$

Phone: +91 844-844-0102

$a[1] \rightarrow *(\alpha + 1) = 5008$

$a[2] \rightarrow *(\alpha + 2) = 5016$

Type : Ext pointers

$\alpha + 2 = 3rd \text{ row}$

$*(\alpha + 2) = 1st \text{ element of } 3rd \text{ row}$

$a[i][j];$

$a[i] \rightarrow *(\alpha + i) \rightarrow 1D \text{ array}$

$*(\alpha + i) + j$

\downarrow

$a[2][3];$

1D pointer to the i^{th} row : $*(\alpha + i)$

$*(\alpha + i)$: integer pointer to the 0^{th} column of i^{th} row

$*(\alpha + 2) = 5016$

: integer pointer not array pointer

$$*(\alpha + 2) + 3 = 5016 + 3 * \text{sizeof}(int)$$

$$= 5016 + 3 * 2 = 5022$$

$*(*(\alpha + 2) + 3)$ find value of this.

\downarrow
34

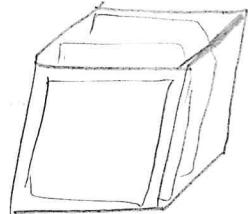
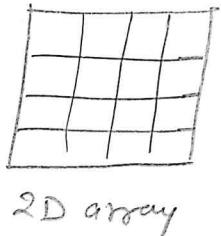
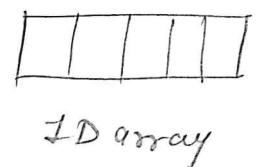
$$*(5022) = 34$$

$\text{int}(*\text{Ptr})[4];$ // Ptr is a pointer to an integer array of size 4.

$\text{Ptr} = a;$ // Creating

a 1D pointer, (pointing to 1D array)
 $(\text{Ptr} + i);$ // 1D integer array pointer

Mail: gatecse@appliedroots.com



3-D array (Multidimensional)

int a [3] [4] [5]



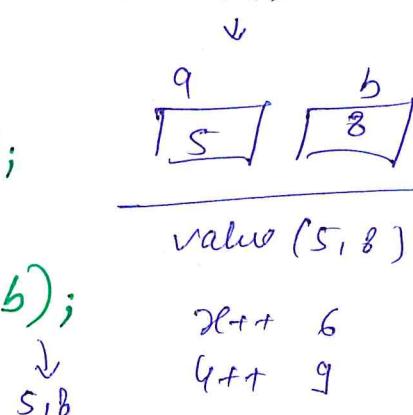
a represents pointer
to 2D array

[9.6] Pointers and Functions

```
#include <stdio.h>
void value (int x, int y);
int main (void)
{
    int a = 5, b = 8;
    printf ("a=%d, b=%d", a, b);
    value (a, b);
    printf ("a=%d, b=%d\n", a, b);
    return 0;
}
void value (int x, int y)
{
    x++;
    y++;
    printf ("x=%d, y=%d\n", x, y);
}
```

O/P = a=5 b=8
a=6 b=9
a=5 b=8

main ()



value (5, 8)

x++ 6

y++ 9

x=6, y=9

after end of value

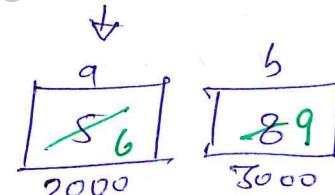
- 11
- ⇒ In call by value, value of actual arg copied to the formal arg, and if formal arg are modified.
 ⇒ Whatever modifications are made in formal args that doesn't reflect back to calling function.

Pass by reference : - In pass by reference, we use pointers.

```
#include <stdio.h>
Void ref (int *p, int *q);
Int main (void)
{
    Int a=5, b=8;
    Printf ("a=%d, b=%d\n", a, b);
    ref (&a, &b);
    Printf ("a=%d, b=%d\n", a, b);
    return 0;
}
Void ref (int *p, int *q)
{
    (*p)++;
    (*q)++;
    Printf ("*p=%d, *q=%d\n", *p, *q);
}
```

By using address, we are modifying value.

main ()



ref (2000, 3000)



$$\begin{aligned}
 &= (*p)++ = \\
 &= (2000)++ = 6 \\
 &(*q)++ = q
 \end{aligned}$$

① return (expr); → returns one value

We want to return multiple values → f (sq, ft, loc)
 ↳ price, tax

call by reference

Phone: +91 844-844-0102

C → address → Pointers

java does not have
concept of pointers

main()

{
} f();f (50.0, loc, float *price, float *tax)
{
} ≡
{

Pointer is also a data-type.

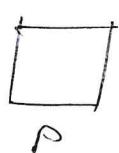
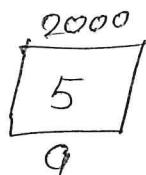
② function returns pointer

int *f (int x, int y)
{
} ≡
{

Syntax:

```
type *f ( )  
{  
} ≡  
{
```

main()	int *f()
{	{
int *P;	int a=5;
P=f();	return &a;
→ a	}

Can not write
local variable

local variable for func' f

Do not return address
of a local variableaddress of integer is integer
pointerfunction will return
value as 2000

So P=2000

Mail: gatecse@appliedroots.com

③ Passing Arrays to a function. +91 844-844-0102

118



main()

```
{
    int a[5] = {1, 2, 3, 4, 5};
    f(a);
}

void f(int a[5]) or int a[5]
{
    a[0]++;
}
```

main()

a : const int pointer

④ Passing 2D arrays to a function:

void f(int a[3][4]) {} ✓

void f(int a[][4]) {} ✓

void f(int a[][]) {} X // left most braces can
not leave empty

void f(int (*a)[4]) {}

void f(int a[][][]) {} X // only left most
can be left empty
Can not leave empty

$a[i][j]$ or $*(*(a+i)+j)$

Mail: gftcecs@appliedroots.com

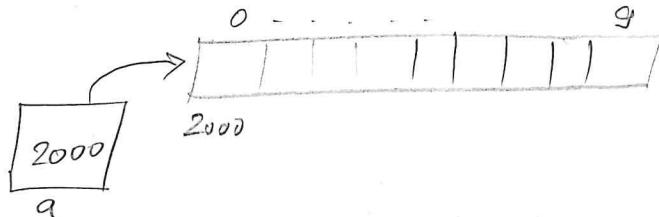
[9.7] Array of pointers and void pointers +91 944-844-0102



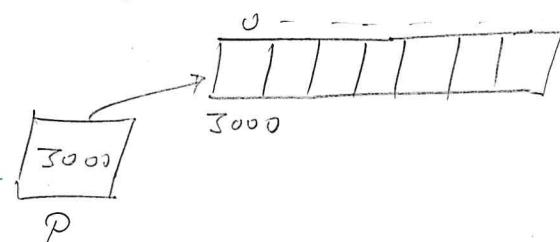
`int a[10]; // array of 10 integers`

pointer → data type

`int *P[10];`



Creating integer pointers array of size 10



`int (*P)[10]; // pointers to an integer array of size 10`

WAP for Array of pointers.

#include <stdio.h>

int main(void)

{

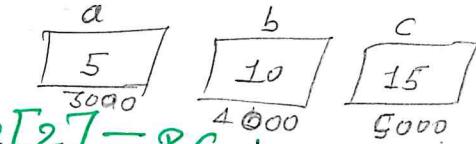
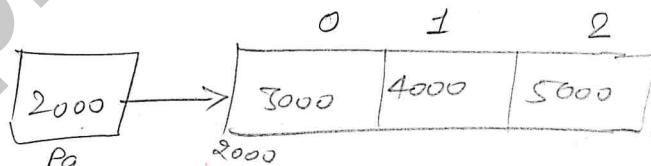
int *Pa[3]

int i, a = 5, b = 10, c = 15;

Pa[0] = &a; Pa[1] = &b; Pa[2] = &c;

for (i = 0; i < 3; i++)

{



`printf("Pa[%d]=%p\n", i, Pa[i]);`

`printf("*Pa[%d]=%d\n", i, *Pa[i]);`

? returns 0;

O/P: Pa[0] = 0x — CO. *Pa[0] = 5

Pa[1] = 0x — C4 *Pa[1] = 10

Pa[2] = 0x — C8 *Pa[2] = 15

2D - Array

Phone: +91 844-844-0102

int $a[3][4] = \{ \{10, 11, 12, 13\}, \{20, 21, 22, 23\}, \{30, 31, 32, 33\} \}$

P[0]	-	<table border="1"> <tr> <td>5000</td><td></td><td></td><td></td></tr> <tr> <td>10</td><td>11</td><td>12</td><td>13</td></tr> <tr> <td>5008</td><td></td><td></td><td></td></tr> <tr> <td>20</td><td>21</td><td>22</td><td>23</td></tr> <tr> <td>5016</td><td></td><td></td><td></td></tr> <tr> <td>30</td><td>31</td><td>32</td><td>33</td></tr> </table>	5000				10	11	12	13	5008				20	21	22	23	5016				30	31	32	33
5000																										
10	11	12	13																							
5008																										
20	21	22	23																							
5016																										
30	31	32	33																							
P[1]																										
P[2]																										

$$a[i][j] \Leftrightarrow *(\text{P}[i]+j)$$

$a[i][j]$
ith row jth col

$$a[1][2] = 22$$

$$*(\text{P}[1]+2)$$

$$\begin{aligned} &+ \\ &5008 + 2 \times \text{sizeof}(int) \\ &5008 + 4 = 5012 \end{aligned}$$

void pointer :

⇒ void pointer means
I don't know how
many bytes to access

* Vptr

Type Cast :

```
void *Vptr;
int *iptr;
float *fptr;
int i = 10;
float f = 1.0;
iptr = &i
fptr = &f
```

$*((\text{int} *) \text{Vptr}) \rightarrow 2 \text{ Bytes}$

↑
explicit type cast
Stacking from
address 2000

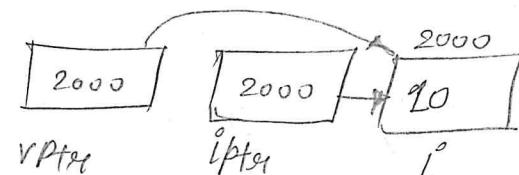
iptr = fptr; X

Copying floating type pointers
in integer pointer.

Vptr = iptr; ✓

Vptr = fptr; ✓

*Vptr Dereferencing



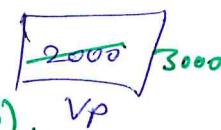
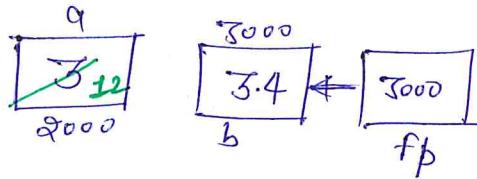
WAP for dereferencing a void pointer: +91 844-844-0102

```
#include <stdio.h>
int main()
{
    int a = 3;
    float b = 3.4, *fp = &b;
    void *vp;
    VP = &a;
    printf("value of a = %d\n", *(int *)vp);
    * (int *)VP = 12;
```

```
printf("value of a = %d\n", *(int *)VP);
    VP = fp;
```

```
printf("value of b = %.f\n", *(float *)VP);
    return 0;
```

}

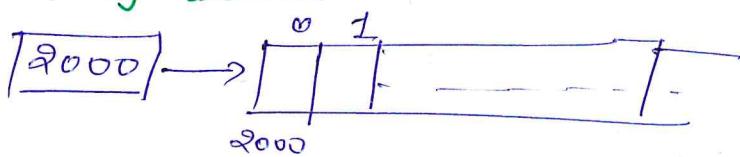


O/P:

```
value of a = 3
value of a = 12
value of b = 3.400.000
```

[9.8] Dynamic Memory Allocation (Pointers)

⇒ `int a[100];` Static Memory allocation



⇒ Instead of storing 100 integer, I am just inserting 30 integers & this decision is taken at run time.

⇒ We can not do this as memory for 100 variables is already created and we can not change it.

⇒ Mail: gatcse@appliedroots.com

⇒ We can not change this allocation dynamically.

⇒ Dynamically or run time **Phone Size + 164 - 44 = 102**

↳ Create
↳ Change

↓
Going to be stored in memory area : heap

- There are some functions which are used for creating dynamically :

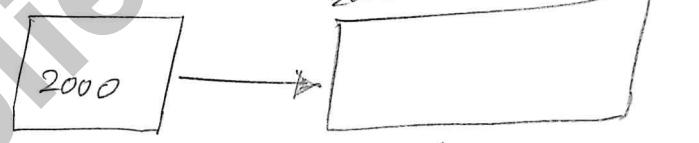
malloc
calloc
realloc
free

`int *P = (int *) malloc(10);`

`void * malloc(Size + size)`

User defined

No. of bytes



`int *P = (int *) malloc(5 * sizeof(int));` → No handle is associated with dynamically created memory.

NULL is a constant which is 0, and defined in stdlib.

`if (P == NULL)`

{

`printf("In Not sufficient memory");`

}

⇒ If no memory is left then it will return NULL.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *P, n, i;
```

printf("enter the no. of integers to be entered:");
 scanf("%d", &n);

P = (int*) malloc(n * sizeof(int));
 if (P == NULL)

{
 printf("Memory not available\n");
 exit(1);
}

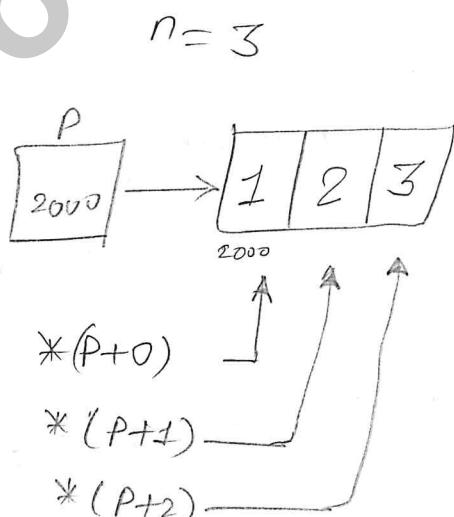
for (i=0; i<n; i++)

{
 printf("enter an integer:");
 scanf("%d", P+i);
}

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

printf("%d\t", *(P+i));

return 0;



O/P : enter the no. of integers to be entered: 3

enter an integer: 1

enter an integer: 2

enter an integer: 3

1 2 3

② Calloc

Same as

$\text{int } *p = (\text{int } *)\text{calloc}(5, \text{sizeof}(\text{int}))$

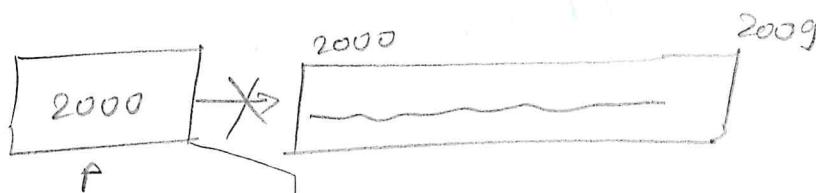
↑
 No. of blocks
 ↑
 size of a block

$(\text{int } *)\text{malloc}(5, \text{sizeof}(\text{int}))$

- ⇒ Calloc also returns NULL if there is insufficient memory.
- ⇒ When you use malloc → It is initialized with some garbage or random value.
- ⇒ When you use calloc → All locations are initialized with zero.

③ Realloc

$\text{int } *p = (\text{int } *)\text{malloc}(10);$

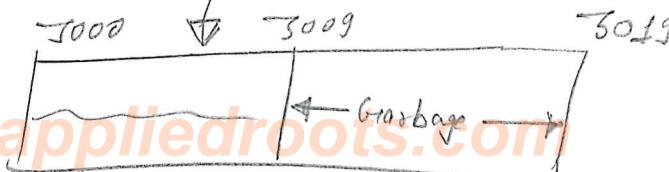


$\text{int } *p = (\text{int } *)\text{realloc}(p, 20)$

- ⇒ New size must be either greater or less than older created memory.

Newsize > Oldsize

Newsize < Oldsize



⇒ If [New size > Old size]. There will be no loss as everything will be copied to newly created memory.

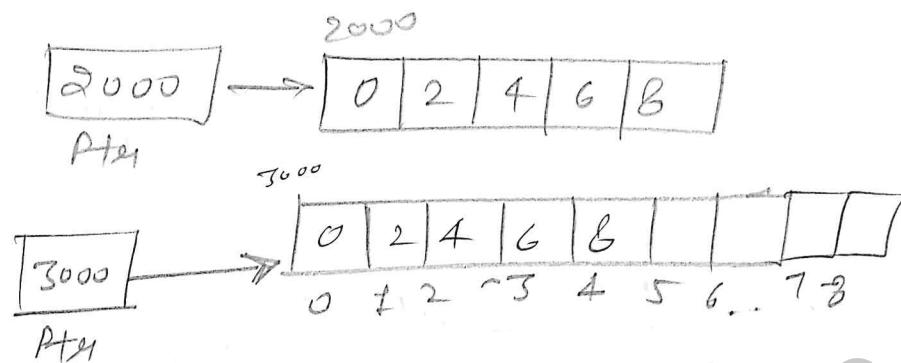
⇒ If [New size < Old size]. There will be loss of data in newly created data as it can only store lesser data.

WAP for realloc function

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i, *ptr;
    ptr = (int *) malloc (5 * sizeof(int));
    if (ptr == NULL)
    {
        printf ("Memory not available\n");
        exit(1);
    }
    for (i=0; i<5; i++)
        *(ptr+i) = i*2;
    ptr = (int *) realloc (ptr, 9 * sizeof(int));
    if (ptr == NULL)
    {
        printf ("Memory not available\n");
        exit(1);
    }
    for (i=5; i<9; i++)
        *(ptr+i) = i*10;
```

for($i=0$; $i < 9$; $i++$) Phone: +91 844-844-0102
 printf("%c.%d", *(ptr+i)),
 return 0;

O/P:



0 2 4 6 8 50 60 70 80

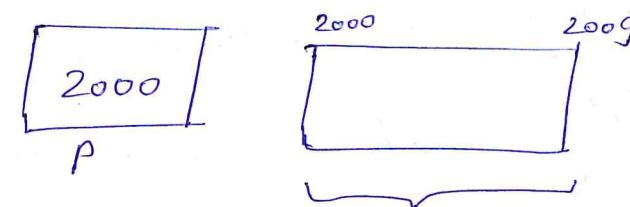
$\text{ptr}[3] \rightarrow *(\text{ptr} + 3)$

Free()

```
void free(void *P);
int *P = (int *)malloc(10)
free(P);
```

\Rightarrow When program terminate
 If

all var will be freed



freed

\Rightarrow It's important to free memory so that other variables can use it

DAM \rightarrow heap - heap will become full.

```

f()
{
    int *p = (int *) malloc (10);
}
main()
{
}

```

Phone: +91 844-844 0102

I cannot use `p` again
bcz scope of `p` is
with in this fn only.

If `p` is not needed
in main function then
we should always
free this.

Memory leak → I have a chunk of memory, but
I don't have any way to access
this memory.

Chapter - 10 Strings

10.1 Introduction to Strings in C

⇒ Strings are nothing but character arrays

String const → String literals "abc" .. " "

Zero or more chars

empty ↓
Nothing

"A" ≠ 'A'
↓
String ↓
 char

a	b	c		\0
---	---	---	--	----

 → String terminator or Null character

to represent end of any string
We store \0 (backslash zero) at the
end of string
Ascii value of \0 ⇒ 0

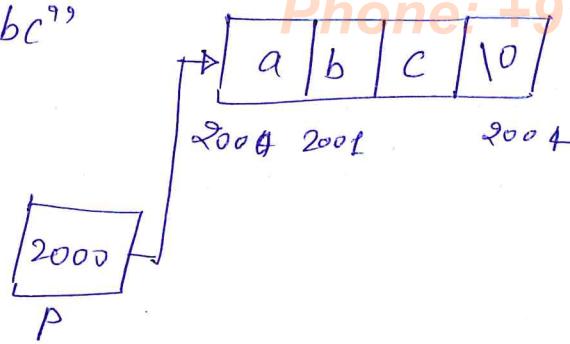
A		\0
---	--	----

Main: gatecse@appliedroots.com

Char *P = "abc"

Phone: +91 844-844-0102

APPLIED
ROOTS

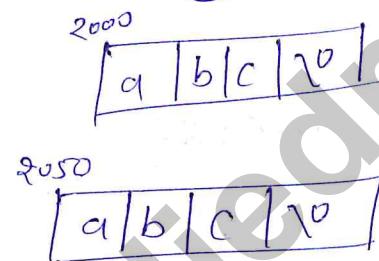


⇒ Internally a pointer is used to access the string constant.

Pointf ("Hi there:\n"),

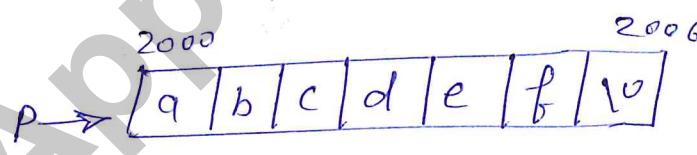
Pointf (char *)

$\begin{array}{c} \leftarrow \\ \text{"abc"} \\ \equiv \\ \text{"abc"} \end{array}$



"abcdef"[2]

↓

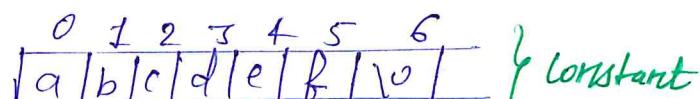


2000 + sizeof(char)

↓
C

Char *P = "abcdef"

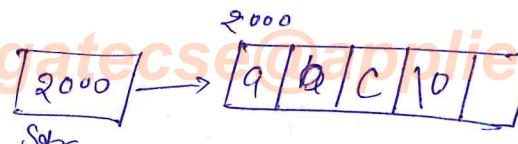
P[1] = a



b = 9

} undefined in
C

Char Str[5] = "abc"; // String initialization



When you are using a string constant as part of a string initialization

In this case, char str[5] = "abc" there is no explicit memory is created.

firstly character array of size 5 is initialized.
String constant not creating new memory.

Escape Character \

"abc\\def" → abc\def

abc"def" → "abc\"def"

↓
telling compiler it is not closing
of double quote.

"abc" "def" → "abcdef" a|b|c|d|e|f|\0|

⇒ String constant is just like character constant.

String Variables :

Char S[] = { 'a', 'b', 'c', 'd', 'e', 'f', '\0' } ;

Same { S → a|b|c|d|e|f|\0|

Mail: gatcse@appliedroots.com

Char S[] = "abcdef"; Char S[10] = "abcdef";

#WAP to print characters of string and address of each character.

```
#include <stdio.h>
int main()
{
    char str[] = "GATE",
    int i;
    for (i = 0; str[i] != '\0'; i++)
    {
        printf("Character = %c\n", str[i]);
        printf("Address = %p\n", &str[i]);
    }
    return 0;
}
```

Str → 

O/P:

Character = G
Character = A
Character = T
Character = E

Address = 0x - - 50
Address = 0x - - 51
Address = 0x - - 52
Address = 0x - - 53

%c for char , %d for int
%f for float , %s for string

} exactly 1B
char is taking 1B only.

#WAP for I/P & O/P strings using scanf() and printf().

```
#include <stdio.h>
int main()
{
    char name[40];
    printf("enter a name");
    scanf("%s", name);
    printf("%s %s", name, "course");
    return 0;
}
```

O/P: Enter a name abc
abc course

[10.2] Library Functions

Platform: in 91 84 reference 844-0102

www.cplusplus.com

<string.h>

<math.h> → sqrt

Stolen

size_t stolen (const char *str); fn declaration

Returns the length of C string str.

char mystr[100] = "test string"; |+|e|s|+| |s|t|r|i|n|g|/|0|

defines an array of characters with a size of 100 chars but the C string with which mystr has been initialized has a length of only 11 characters.

∴ While sizeof(mystr) evaluates to 100

stolen (mystr) returns 11

```
#include <stdio.h>
#include <string.h>
int main()
{
    char szInput[256];
    printf("Enter a sentence:");
    gets(szInput); // scanf("%s", szInput);
    printf("The sentence entered is %u characters long.\n",
        (unsigned) stolen(szInput)),
    return 0;
}
```

O/P: Enter a sentence: just testing

= 10. random 1 1 .. in another para

Mail: gatecse@appliedroots.com

```

#include <stdio.h>
#include <string.h>
unsigned int astrlen (char str[])
{
    int i=0;
    while(str[i]!='\0')
        i++;
    return i;
}

int main()
{
    char str[50];
    printf("enter a string:");
    gets(str);
    printf("Length of string is: %u\n", astrlen(str));
    return 0;
}

```

0	1	2	3	4	5	6
a	b	c	d	e	f	\0

O/P: Length of the string is : 6

strcmp

Compares the C String Str1 to the C String Str2.

int strcmp (const char * Str1, const char * Str2);

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until the terminating null-character is reached.

33

⇒ This function performs a binary comparison of the characters. For a function that takes into account locale-specific rules.

Phone: +91 844-844-0102

Ex:

“abcdef” “abdef”

< 0

0

> 0

“abdef” “abcd”

```
#include < stdio.h >
```

```
#include < string.h >
```

```
int main()
```

```
{
```

```
char key[] = "apple";
```

```
char buffer[80];
```

```
do {
```

```
    printf("Guess my favorite fruit?");
```

```
    fflush(stdout);
```

```
    scanf("%s", buffer);
```

```
    while (strcmp(key, buffer) != 0);      apple, orange
```

```
    puts("Correct answer!");
```

```
    return 0;
```

```
}
```

user enter : orange

O/P:

Guess my favorite fruit? orange

Guess my favorite fruit? apple

Correct answer!

Mail: gatcse@appliedroots.com

$\text{char *strcpy} = (\text{char *dest}, \text{const char *source});$

- ⇒ Copies the C string pointed by source into the array pointed by destination. Including terminating null character (and stopping at that point)
- ⇒ To avoid overflows, the size of the array pointed by destination shall be long enough to contain the same string as source (including terminating null character), and should not overlap in memory with source.
- ⇒ Destination is returned.

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str1 [] = "sample string",
        str2 [] = [40],
        str3 [] = [40];
    strcpy (str2, str1);
    strcpy (str3, "Copy successful");
    printf ("str1: %s\n str2: %s\n str3: %s\n", str1, str2, str3);
    return 0;
}
```

When not assigning,
we need to use strcpy for
assigning.

O/P:

Str1: Sample string

Str2: Sample string

Str3: Copy successful

Str2[0] = 'a' ✓, Str1[1] = 'b', ✓

Str2[0] = "abcdef", X Not valid

Mail: gntcece@appliedroots.com

~~StrCat~~ : for concatenation

~~APPLIED
ROOTS~~

① ~~Append + 01814-941012~~
Copies a copy of source string to the destination string.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[80];
    strcpy(str, "these");
    strcat(str, "strings");
    remove 'o' and add strings
    strcat(str, "are");
    strcat(str, "concatenated");
    puts(str); // Puts means print
}
```

these10
remove 'o' and add strings10

char * strcat (char * dest, const char * source),

for, while, arrays, pointers

- ② terminating null character in destination is overwritten by the first character of source.
- ③ Null character is included at the end of the new string formed by the concatenation of both in the destination.

StringyAPPLIED
ROOTS

`char * strncpy (char * destination, const char * source, size_t num)`

- ⇒ copies the first num characters of source to dest.
- ⇒ If the end of source C String (which is signaled by a null character) is found before num characters have been copied.
- ⇒ Destination is padded with zeros until a total of num characters have been copied. Written to it.
- ⇒ No Null-character is implicitly appended at the end of destination if source is longer than num. Thus in this case destination is not considered a null terminated C String (reading it as such would overflow).
- ⇒ destination and source shall not overlap.

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char Str1[] = "To be or not to be";
    char Str2[40];
    char Str3[40];
    strncpy (Str2, Str1, sizeof(Str2));
    strncpy (Str3, Str2, 5);
    Str3[5] = '0';
    puts (Str1);
    puts (Str2);
    puts (Str3);
}
```

O/P: To be or not to be

To be or not to be

To be

**APPLIED
ROOTS**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[20];
    char str2[20];
    strcpy(str1, "To be");
    strcpy(str2, "or not to be");
    strcat(str1, str2, 6);
    puts(str1);
    return 0;
}
```

O/P: To be or not

Strstr

```
const char * strstr (const char * str1, const char * str2);
char * strstr (char * str1, const char * str2);
```

Locate substring

Returns a pointer to the first occurrence of str2 to str1, or a null pointer if str2 is not part of str1.

The matching process does not include the terminating null-characters, but it stops there.

```
char * strstr (const char *, const char *);
```

Mail: gatecse@appliedroots.com

```

#include <stdio.h>
#include <string.h>
int main()
{
    char str[] = "this is a simple string";
    char *pch;
    Pch = strstr(str, "simple");
    strcpy(pch, "sample", 6);
    puts(str);
    return 0;
}

```

Simple
↓
sample

Copy only 6 characters

O/P: This is a sample string

Word

Search + Replace

F R

Strtok

String tokenizer

, space

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{

```

Char str[] = "This is a sample string";
 Char *pch;

Printf("Splitting string | %s | into tokens: \n", str);

Pch = strtok(str, ",.-");

Pch → This
Pch → is

printf("%s\n", pch);

Pch → a

Pch = strtok(NULL, ",.-");

return 0;

O/P: Splitting String "This, a sample string." into tokens:
 This
 a
 sample
 string

Sprintf

Write formatted output to sized buffer

```
#include <stdio.h>
int main()
{
    char buffer[50];
    int n, a=5, b=3,
    n = sprintf(buffer, "%d plus %d is %d", a, b, a+b);
    printf("%s [%d] is a string %d characters long", buffer, n);
    return 0;
}
```

SScanf : Instead of taking input from keyboard, take it from other strings.

MAP to convert strings to integer and float values.

```
#include <stdio.h>
int main()
{
    char str1[10] = "1000";
    char str2[10] = "125.50";
    int x; float y;
    sscanf(str1, "%d", &x);
    sscanf(str2, "%f", &y);
    printf("Value of x=%d, value of y=%f", x, y);
    return 0;
}
```

O/P: value of $x = 1000$, value of $y = 125.50$

Phone +91 844-844-0102



→ covered main major functions of string.h

<math.h> functions math.h c reference

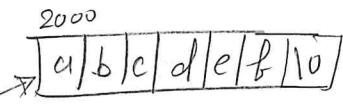
- ① double acos (double x)
- ② double asin (double x)
- ③ double atan (double x)
- ④ double atan2 (double x)
- ⑤ double cos (double x)
- ⑥ double cosh (double x)
- ⑦ double sin (double x)
- ⑧ double sinh (double x)
- ⑨ double tanh (double x)
- ⑩ double exp (double x)
- ⑪ double frexp (double x, int * exponent).
- ⑫ double ldexp (double x, int exponent)
- ⑬ double log (double x) (base e)
- ⑭ double log10 (double x)
- ⑮ double modf (double x, double * integer)
- ⑯ double Pow (double x, double y);
- ⑰ double sqrt (double x)
- ⑱ double ceil (double x)
- ⑲ double fabs (double x) return absolute of x
- ⑳ double floor (double x)
- ㉑ double fmod (double x, double y)

Pointers, Arrays and Strings

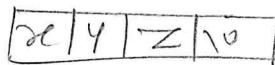
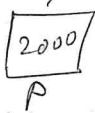
From +91 844-844-0102



Char *P = "abcdef";

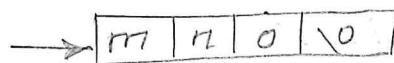


Char S[] = "xyz";

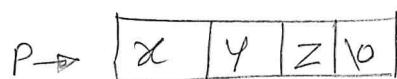


Stg S[7] S[2] S[5]

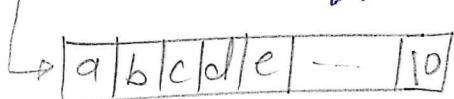
Char = "mno";



P = "xyz";



P = "abcdefghijkl"

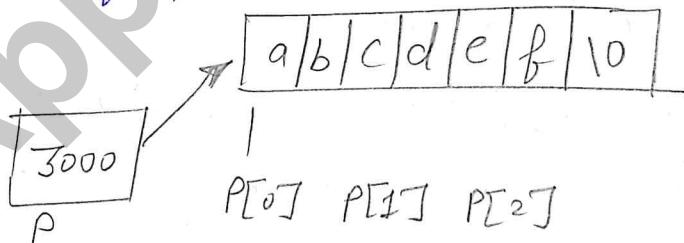


scanf("%s", S);

abc ✓
abcd ✗

→ Char *P = "abcdef";

P[2]=q;



scanf("%s", P),

strcpy(P, "abc"); ✗ Not allowed bcoz we are trying to put const

Char *P;

scanf("%s", P);

P = (char *)malloc(20);

Array of Strings

Phone: +91 844-844-0102

Char $a[5][10] = \{ "White", "Red", "green", "blue", "yellow" \}$

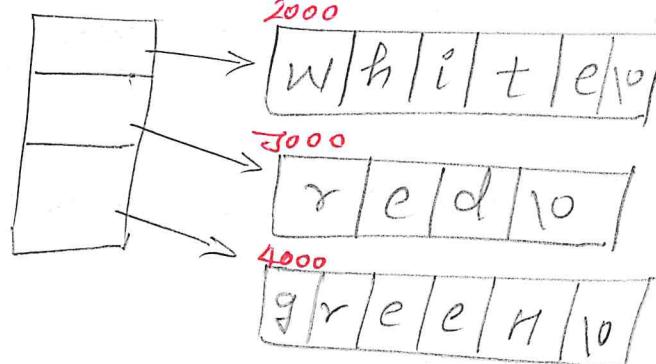
	0	1	2	3	4	5	6	7	8	9
$a[0] \rightarrow 0$	W	H	I	t	e	\0				
$a[1] \rightarrow 1$	R	e	d	\0						
$a[2] \rightarrow 2$	g	r	e	e	n	\0				
$a[3] \rightarrow 3$	b	l	u	e	\0					
$a[4] \rightarrow 4$	y	e	L	L	\0	\0				

Char $a[5][10] = \{ \{ 'W', 'H', 'I', 't', 'e' \},$
 $\quad \quad \quad \{ 'R', 'e', 'd' \},$
 $\quad \quad \quad \{ 'g', 'r', 'e', 'e', 'n' \},$
 $\quad \quad \quad \{ 'b', 'l', 'u', 'e' \},$
 $\quad \quad \quad \{ 'y', 'e', 'L', 'L', '\0' \} \}$

Array of Pointers to Strings

Char *AP[5] = { "White", "red", "green" };

AP[0]
AP[1]
AP[2]



Char $a[5][10];$

Char *AP[5];

for strings already
Space allowed

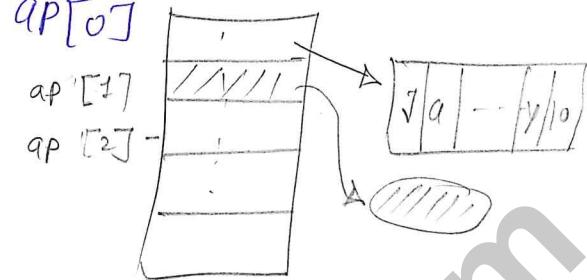
→ space for strings

is not allocated

Mail: gatecse@appliedroots.com

$a[0] = \text{"jan"}$
const

\times in memory
JAN will be
creating
and try to allocate
points to this
memory

 $a[0] = \text{"january"}$
strcpy (a[0], "jan");
ap[0]**ap[1]****ap[2]**
scanf ("%f", a[2]);

take the input & store it

scanf ("%s", ap[2]);

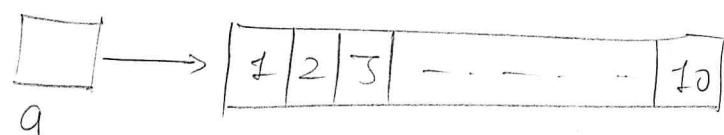
\times As there is not space allocated
for ap[2]

 $ap[2] = (\text{char} *) \text{malloc}(20);$
scanf ("%s", ap[2]);

\Rightarrow If you print "abc" in memory. It is going to print constant

[10.4] Pointers to Array revisited

Int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};



int * const

Mail: **gatecsse@appliedroots.com** value stored at (a+2)

② Pointer to array (1D)

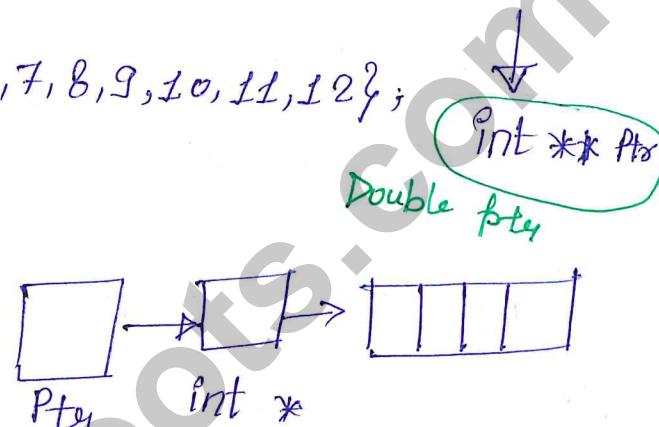
Phone: +91 844-844-0102

`int (*P)[10];`

datatype \rightarrow `int * P[10];` \rightarrow array of int ptrs \downarrow pointer to int

`int a[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

ID	0	1	2	3
(a+1)	1	2	3	4
2	5	6	7	8
	9	10	11	12



as Ptr \rightarrow 1D array

`a` $\xrightarrow{1D}$
(2000)

`a+1` \xrightarrow{ID}
2008

`(a+2)` \xrightarrow{ID}
2016

$*(\text{a}+1) \Rightarrow \text{int *} = 2008$

$*(\text{int} *) \Rightarrow \text{int} *$

$\text{int} * \Rightarrow \text{value}$

$$a[1][0] = *(*(\text{a}+1)+0)$$

$$a[i][j] = *(*(\text{a}+i)+j)$$

Consider a Program.

Phone: +91 844-844-0102

#include <stdio.h>

int main()

{

int a[3][4][2] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20};

printf("%d", *(*(a+2)-3)[2]);
return 0;

}

What is output of code.

Soln,

\Rightarrow

$a+0$	
1	2
3	4
5	6
7	8

a is pointer
to 2D array

$a+1$	
9	10
11	12
13	14
15	16

$a+2$	
17	18
19	20
-	-
-	-

① $a \rightarrow$ address of (1)

② $a+2 \rightarrow$

③ $*(\mathbf{a}+2) \rightarrow$ pointer to 1D array $= a[2]$
int **

④ $(*\mathbf{a}+2)-3 \Rightarrow$ pointer to 1D array

Travers back \rightarrow address of (11)

⑤ $*(\mathbf{*(a+2)-3})[2] \equiv *(\mathbf{(*a+2)-3}+2)$

$\equiv *$ (Pointer to 1D array + 2)
2 row

What is output of C code.

Phone: +91 844-844-0102

14e

```
#include <stdio.h>
int main()
```

```
{  
char arr[2][3] = { 'A', 'P', 'P', 'L', 'E', 'E', 'D',  
'C', 'O', 'U', 'R', 'S', 'E' };
```

char (*P)[2][3] = arr;

Printf("%d", (*(*P)[0] + 5)) - (*(*P)[1] - 3)).
between 0;
{

Solⁿ

*P[0]		
A	P	P
L	I	E
D		C

*P[1]		
O	U	R
S	E	Y
Y	Y	Y

*P = Pointer to 2D

*P[0] = *(*(P) + 0)

(*P)[0] + 5 - *(*P)[1] - 3

B - D = I

Ptr to 1D array → int **

Ptr to 2D Array → int ***

Ptr to 3D Array → int ****

Dereferencing

= * (int ***)

= (int **)

[12.1] Structures & Unions

Array: collection of similar items → homogeneous

```
int a[10];
```

Student → logical entity/object (real-world)

Harrue, rno, marks
 String int float

Struct Student {
 Keyword Identifier

char name [30];
 int rno;
 float marks;
 } ; S1, S2, S3;

} members/attributes
 Collection of the heterogeneous members

all of them are attributes of a real world entity.

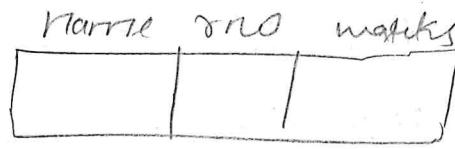
⇒ Declare a variable:

① with no definition

② Struct Student S1, S2, S3;

Memory

Struct Student S1;



⇒ all attributes or members of struct are stored contiguously

Initialize Structure :

Phone: +91 844-844-0102

(1) Init along with definition of the struct.

(2) Struct Student S2 = { "Neelima", 1, 98.0 }
 ↑ Decla ↓ Init

(3) Struct Student S3 = { "Naiveen", }; eno & marks
 ↓
 init to 0

(4) Struct Student {

 = float marks = 90.0; X NOT valid
 }

No m/r

Access :

Struct Student S1;

X S1.name = "Murali"

because Murali
is a constant

name	eno	marks
JOB	25	45

strcpy (S1.name, "Murali"); ✓

S1.eno = 2;

S1.marks = 95.0;

↑ dot operator for accessing
elements

MAP to print the values of structure member

```
#include <stdio.h>
#include <string.h>
```

Struct student

```
{  
    char name[20];  
    int rollno;  
    float marks;  
};
```

```
int main()
```

```
{
```

```
Struct Student str1 = { "Satisf", 25, 68 };
```

```
Struct Student stu2, stud3;
```

```
strcpy(stud3.name, "Naveen");
```

```
stu2.rollno = 26;
```

```
stu2.marks = 98;
```

```
Printf("Enter name, rollno, and marks for Stud3: ");
```

```
Scanf("%s %d %f", stud3.name, &stud3.rollno, &stud3.marks);
```

```
Pointf("Stud1: %s %d %.2f\n", stud1.name, stud1.rollno,  
stud1.marks);
```

```
Pointf("Stud2: %s %d %.2f\n", stud2.name, stud2.rollno, stud2.marks);
```

```
Pointf("Stud3: %s %d %.2f\n", stud3.name, stud3.rollno, stud3.marks);
```

between %

Assignment :

```
s1 = s2;
```

Struct student

```
int a, b
```

```
a = 10
```

```
b = a;
```

- (1) Unary, relational, arithmetic operations are not allowed
otherwise → Not allowed on struct variables.
- (2) attributes / members of struct s1.rollno ←

Mail: gatcse@appliedroots.com

~~S1 & S2~~ X

~~S1 > S2~~ X ~~PI~~ ~~S1 & S2~~ X 91 84 101 102

~~-S1;~~ X

$\Rightarrow \underline{\text{sizeof}}$

✓ `sizeof (struct Student);`

✓ `sizeof (S1);`

$\Rightarrow \underline{\text{Arrays}}$

`Struct Student sa[10];`

Access →

`sa[0].name;`

`sa[0].rollno;`

`:`
`sa[i].name;`

`int a[10];`

for accessing

`a[0], a[1]`

\Rightarrow `Struct Student a[3] = { { "Veerita", 6, 96 }, { "Satish", 2, 98.0 }, { "Naveen", 3, 99.0 } };`

\Rightarrow It is same as initializing an array.

[12.2] Let's dive into Structures

Phone: +91 844-844-0102

1) Nested Structure: Structure with in struct

Struct student {
 char name[30];
 int sno;
 Struct date {
 int day;
 int month;
 int year;
 } dob;
};
S1, S2;

definition of struct student

⇒ Dot operator has the same precedence as parenthesis

S1.dob.day ⇒ ((S1.dob).day) L to R

(o) dot operator: use to access members of structures.

(→) Arrow operator: is powerful w.r.t pointers to structure.

② Another way of writing

Struct date {
 };
};
;

Struct student {

=

Struct date dob;

=

};
;

2) Pointers to Structures:

Struct student S1, *P;

Name, sno, marks

P = &S1;

S1.Name

(*P).Name

S1.sno

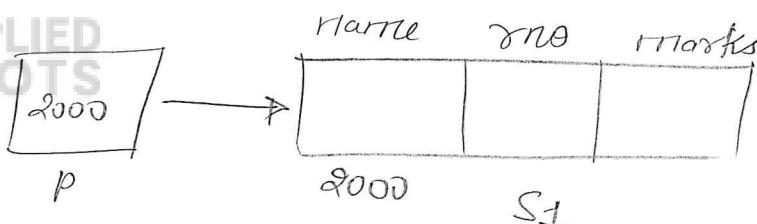
dot has higher precedence over *

*P.Name ⇒ *(P.Name) X

Mail: gatcse@appliedroots.com

If want to access Harne using pointers

Phone: +91 84443140102 ✓



$(*P).rollno$ ✓
Complex Syntax

$$P \rightarrow \text{Harne} \equiv (*P).\text{Harne}$$

Much more neater,
Less error prone
Syntax

(2) ptr to an attribute

$$\text{int } *P_{\text{tr}} = \& ST.\text{rollno};$$

(3) Pointer Inside Structure : ($\&$, $*$, $.$, \rightarrow operators)

Struct Student {

 char name[20];

 int *P;

} ST, *SP;

↓
Student
Structure
variable

↳ pointer to the Student
Structure

$$SP = \& ST; \quad \checkmark$$

$$S.P$$

$$*S.P = *(S.P) \quad \text{dot has higher precedence}$$

$$*SP \rightarrow P \Rightarrow *(SP \rightarrow P)$$

WAPP to understand pointers to structures.

Phone: +91 844-0102

```
#include <stdio.h>
```

```
Struct Student
```

```
{
```

```
    char name[20];
```

```
    int rollno;
```

```
    int marks;
```

```
};
```

```
int main()
```

```
{
```

```
Struct Student stud = { "Neelima", 25, 68 };
```

```
Struct Student *ptr = &stud; → pointing to address of
```

```
printf ("Name - %s", ptr → name);
```

```
printf ("Rollno - %d", ptr → rollno);
```

```
printf ("Marks - %d", ptr → marks);
```

```
return 0;
```

```
}
```

(4) Structures & Functions :

⇒ Structures are user-defined

(1) Passing Struct members as arguments .

```
#include <stdio.h>
```

```
#include <string.h>
```

```
Struct Student {
```

```
    char name[20];
```

```
    int rollno;
```

```
    int marks;
```

```
};
```

```
void print (char name[], int rollno, int marks);
```

int main()

Mail: gatecse@appliedroots.com

Struct Student stud1 = { "Neelima", +91 844-844-0102
Struct Student stud2;

Stud1 (stud1.name, "Neelima"),
stud1.rollno = 1,
stud1.marks = 90;

Printf (stud1.name, stud1.rollno, stud1.marks);

Printf (stud2.name, stud2.rollno, stud2.marks);

between 0;

}

void print (char name[], int rollno, int marks)

{

printf ("Name - %s\n", name);
printf ("Rollno - %d\n", rollno);

printf ("Marks - %d\n", marks);

}

(2) Pass Structure variable as argument to function.

void print (struct student);

Printf (stud1);

Printf (stud2);

between 0;

void print (struct student stud)

{

-

{

(3) Passing pointers to a Structure as an argument

Phone: +91 944-300-0102

⇒ Can pass structure by reference

```
void print(struct Student *);  
void inc_marks(struct Student *);  
int main()  
{  
    struct Student stud1 = {"Satisf", 1, 87};  
    struct Student stud2 = {"Naveen", 2, 90};  
    inc_marks(&stud1);  
    inc_marks(&stud2);  
    printf("%d", &stud1);  
    printf("%d", &stud2);  
    } between 0;
```

```
void inc_marks(struct Student *studptr)  
{  
    (*studptr -> marks)++; any changes  
made here will be visible  
}  
void print(struct Student *studptr)  
{  
    printf("Name - %s", studptr -> name);  
    printf("Rollno - %d", studptr -> rollno);  
    printf("Marks - %d", studptr -> marks);  
}
```

(4) Return a structure itself

Phone: +91 844-844-0102

+96

#WAP to understand how a structure variable is returned from a function

```
#include <stdio.h>
```

```
struct student
```

```
{
```

```
char name[20];
```

```
int rollno;
```

```
int marks;
```

```
};
```

```
void print(struct student),
```

```
struct student change(struct student)
```

```
int main()
```

```
{
```

```
struct student stud1 = {"Satisf", 20, 87};
```

```
struct student stud2 = {"Naveen", 21, 90};
```

```
stud1 = change(stud1);
```

```
stud2 = change(stud2);
```

```
printf("%s", stud1.name);
```

```
printf("%d", stud1.rollno);
```

```
printf("%d", stud1.marks);
```

```
}
```

```
struct student change(struct student stud)
```

```
{
```

```
stud.marks = stud.marks + 5;
```

```
stud.rollno = stud.rollno - 10;
```

```
return stud; — returning variable of type struct
```

```
void print(struct student stud)
```

```
{
```

```
printf("Name - %s\n", stud.name);
```

```
printf("Rollno - %d\n", stud.rollno);
```

```
printf("Marks - %d\n", stud.marks);
```

WAP to understand how arr of Structures is sent to a function

include <stdio.h>

Struct Student

{

 char name[20];
 int rollno;
 int marks;
};

void print(Struct Student);

void dec-marks(Struct Student stuarr[]);

Ent main()

{
 int i;
 Struct Student stuarr[3] = {

 { "Neelirra", 12, 98 },
 { "Naveen", 11, 97 },
 { "Subbu", 13, 89 }
 };

 dec-marks(stuarr);

 for (i=0; i<3; i++)

 Print(stuarr[i]);

 return 0;

}

void dec-marks(Struct Student stuarr[])

{
 int i;

 for (i=0; i<3; i++)

 stuarr[i].marks = stuarr[i].marks - 10;

 void Print(Struct Student stud)

 Print("Name - %s\n", Stud.name);

 Print("Rollno - %d\n", Stud.rollno);

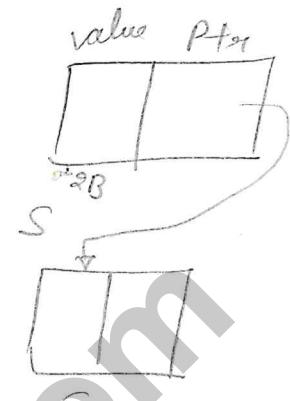
 Print("Marks - %d\n", Stud.marks);

Mail gatooce@appliedroots.com

Struct node {

```
int value;
Struct node * Pte;
} S, S1;
```

S.Pte = &S1;



⇒ self-referential structures are important for Linked-Lists.

[12.3] Unions and typedef

Unions :
+
keyword

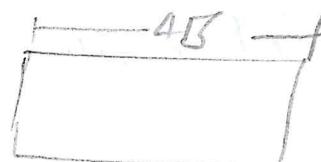
user-defined data type

union result {

```
int marks; - 2B
char grade; - 1B
float percentile; - 4B
}; r;
```

⇒ Create memory of max of these three numbers not 7B, because 4B is (2B, 1B, 4B).

⇒ I can either store marks or grade or percentile. I can't store all 3 like structures.



Number → any numeric
 ↓
 int
 ↓
 long
 ↓
 float
 ↓
 double

Union numbers {

```
int i;
long l;
float f;
double d;
}
```

Phone: +91 844-844-0102

$U.i = 10;$
 $U.l = 100L;$
 $U.f = 10.0f;$
 $U.d = 10.0f;$

#WAP to compare the memory allocated for a union and structure variable.

#include <stdio.h>

Struct stag {

```
char c;
int i;
float f;
};
```

Union Utag {

```
char c;
int i;
float f;
};
```

int main(void)

Union Utag Uvar;

Struct stag Svar;

printf ("Size of Svar = %u\n", sizeof(Svar));

printf ("Address of Svar = %p\n", &Svar);

printf ("Address of members: %p %p %p\n",

&Svar.c, &Svar.i, &Svar.f);

Mail: gatecs@appliedroots.com

Phone: +91 844-844-0102



printf ("size of Uvar = %u\n", sizeof(Uvar));
 printf ("Address of Uvar: %p\n", &Uvar);
 printf ("Address of members: %p %p %p\n",
 &Uvar.C, &Uvar.i, &Uvar.f);
 };

O/P: size of Svar = 12

Address of Svar = 0x --- 700

Address of members = 0x --- 700

0x-8908
708

Size of Uvar = 4

Address of Uvar = 0x --- 6f0

Address of members = 0x --- 6f0

0x-6f0
0x...86f0

how do you Initialize Union :

only initialize the first member of union or first attribute of union. You can not initialize more than that.

Union result res = {89};

```
int main()
{
    Union result
    {
        int marks;
        char grade;
        float per;
    }res;
```

res.marks = 90;

printf ("Marks: %d\n", res.marks);

res.grade = 'A';

printf ("Grade: %c\n", res.grade);

res.per = 85.5;

printf ("Percentage : %d\n", res.per);

- Phone: +91 844-844-0102
- ⇒ Can use union inside struct.
 - ⇒ Unions can be nested.
 - ⇒ Arrays of Unions.
 - ⇒ fn arg & return type in unions just like struct
 - ⇒ pointers to unions.
 - ⇒ Unions can be self referenced.

typedef

It is very useful in structures and unions.

```
typedef int num; // redefining int as num
num a, b; // int a, b;
```

```
Struct student st;
```

```
typedef struct student stud; // user-defined datatype
stud st; // readable /  
           // crisp / clear
           // redefining the  
           // as stud
```

13.1 Introduction to files :

Everything is created stored in the RAM

variable → **RAM**
MM → Also program terminate,
whatever is stored in RAM
is lost

CPU ↔ RAM ↔ disk: Permanent storage of data
HDD
SSD (Solid state drive)
SD-card

There are two methods / modes of storage?

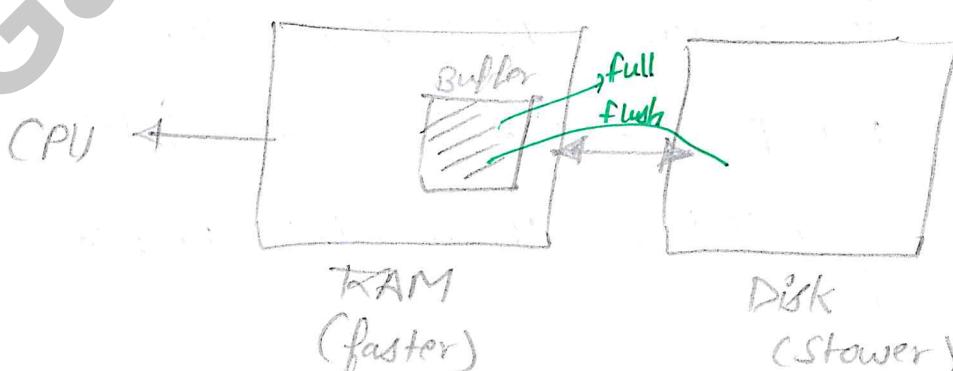
① Text → Human-readable, end of file (ASCII : 26)
Portable (ASCII, unicode)

② Binary → System
RAM → disk
faster
(.dll, .exe)

Ctrl+Z
Ctrl+D

Ex: (My name is reetika) → footer

Buffer? Some space within the RAM (Temporary space)



abcdefg hij k [L]
 ↑↑

Phone: 91 844-844-0102
 efficient

as storing a then b -- if want to store 'l' as buffer is full. It will flush.

⇒ youtube video, which is not yet fully there. It is stored as buffered then you can watch

`fflush()` - file flush in stdio.h

If something is half full. You can ask to flush it forcefully using fflush.

① Step 1 ① opening

② read/write/appending

③ close

① opening :

<stdio.h>

File structure

↳ Name, Status, buffer size, current position

`FILE *fp;` `#include <stdio.h>`

`typedef struct FILE;`

`fp = fopen("file1n", "w");`

↑
 name of file FILENAME_MAX

full path - C:\Documents\file1.txt

text

"w" - Want to open my file in Write mode.

a - append

r - read



"wt" - write + read (previous data is erased)

"r+" - read + write (not erased, append)

Binary

ce Hb^{99} binary

“ab” append mode binary file

ERRORS

```
fP = fopen ("efile1", "ew");
```

$f_p == \text{NULL}$ → error → No space on disk
No permission {Linux}
doesn't exist file or a mode

Closing a FILE

`int fclose (FILE *fp)`

↳ all the file buffer has flushed \rightarrow disk
buffers are freed.

between 0 (success)

Written by OS & —
Controlled by Programs —
—1 (EOF) stdio.h
↑
Error
↑
const

Stdin → keyboard

Stdout → monitor

std::err → monitor

predefined const FILE * in
stdio.h

`int fcloseall(void)` → all the file pointers in
main() program are closed

FILE *fp = fopen(“”, “”);
{ =

16

```
=  
fclose(fp); // all the buffers are flushed  
// clearly loss if don't write .
```

```

#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fptr; // creating file pointer
    int ch;
    if ((fptr = fopen("test", "w")) == NULL)
    {
        printf("File doesn't exist\n");
        exit(1);
    }
    printf("Enter text : \n");
    while ((ch == getchar()) != EOF) // getting character
        fputc(ch, fptr); // writing character to keyboard
    fclose(fptr);
    return 0;
}

```

Ctrl+Z or Ctrl+D

int fputc(int char, FILE *stream)

⇒ Writes a character (an unsigned char) specified by the argument char to the specified stream, and advances the position indicator for the stream.

Parameters:

Char: character to be written. This is passed as its int promotion.

Stream: pointer to a file object that identifies where the character is to be written.

Mail: gatecse@appliedroots.com

Return value : If there are no error, some characters that has been written is returned.

If an error occurs, EOF is returned and error indicator is set.

```
#include <stdio.h>
int main()
{
    FILE *fp;
    int ch;
    fp = fopen("file.txt", "w+");
    for(ch = 33, ch <= 100; ch++) {
        fputc(ch, fp);
    }
    fclose(fp);
    return 0;
}
```

WAP to get the use of fgetc()

#include <stdio.h> get character from file

#include <stdlib.h>

int main()

{ FILE *p;

char c;

if ((p = fopen("test", "r")) == NULL)

{ printf("error in opening file\n");

exit(1);

while ((c = fgetc(p)) != EOF)

printf("%c", c);

fclose(p);

} return 0;

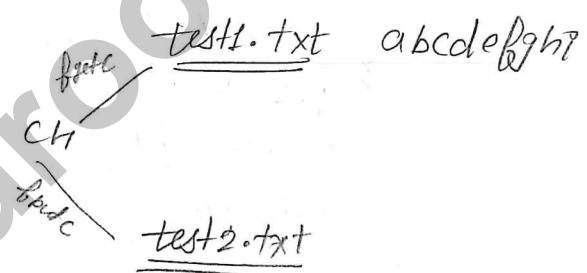
`int fgetc(FILE *Stream)` : gets the next character
(an unsigned char) from the
specified Stream

⇒ advances position indicator for the stream.

abcde^fgghi
↑
read one character
2 advances posn

how fgetc and fputc can be used. ⇒ If want to copy
one file to other.

```
int main()
{
    FILE *sptr, *dptr;
    char ch;
    if ((sptr = fopen("test1.txt", "r")) == NULL)
    {
        printf("Error in opening source file\n");
        exit(1);
    }
    if ((dptr = fopen("test2.txt", "w")) == NULL)
    {
        printf("Error in opening destination file\n");
        exit(1);
    }
    while ((ch = fgetc(sptr)) != EOF)
        fputc(ch, dptr);
    fclose(sptr), fclose(dptr);
    return 0;
}
```



#WAP to understand ~~fputs~~ (read strings) +91-844-844-0102

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fptr;
    char str[80];
    if ((fptr = fopen ("test", "w")) == NULL)
    {
        printf ("Error in opening file\n");
        exit(1);
    }
    printf ("Enter the text\n");
    printf ("To stop entering, press Control+d/ctrl+z\n");
    while (gets(str) != NULL)
        fput (str, fptr);
    fclose (fptr);
}
return 0;
```

reads from std I/O/keys
Press enter - for going to next line

#WAP to understand fgets

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fptr;
    char str[80];
    if ((fptr = fopen ("test", "r")) == NULL)
    {
        printf ("Error in opening file\n");
        exit(1);
    }
    while (fgets (str, 80, fptr) != NULL)
        puts (str);
    fclose (fptr);
}
return 0;
```

length of varie & file pointer
Size at most 80

fgets

`char *fgets(char *str, int n, FILE *stream)`

⇒ Reads a line from the specified stream and store it into the string pointed to by str.

It stops when either (n-1) characters are read, the new line character is read, or the end-of-file is reached, whichever comes first.

fputs

`int fputs(const char *str, FILE *stream)`

Write a string to a specified stream up to but not including NULL character.

printf
↳
stdout

scanf
↳
stdin

printf → formatted file I/O
sprintf sscanf

WAP to understand the use of fprintf()

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    FILE *fp;
    char name[10];
    int id;
    if ((fp = fopen("test", "ew")) == NULL)
    {
        printf ("Error in opening file\n");
        exit(1);
    }
```

Mail: gatcse@appliedroots.com
`printf ("Enter your name & id);`

Scarf ("%s %d", name, P2(id)); phone: +91 844-844-0102

fprintf(fp, "My name is %s and id is %d", name, id);
fclose(fp); }
return 0; } file pointer
}

fscanf

Keyboard → scanf → var
file → fscanf → var
String → sscanf

#include <stdio.h>

#include <stdlib.h>

struct student {

 char name[20];
 float marks;
}; Stud;

int main()

{ FILE *fp;
if ((fp = fopen("test", "r")) == NULL)

{ printf("Error in opening file\n");

exit(1);

printf("NAME % MARKS\n");

while (fscanf(fp, "%s %f", Stud.name, Stud.marks) != EOF)

printf("%s %f\n", Stud.name, Stud.marks);

fclose(fp);

return 0; }

Block input output

Phone: +91 844-844-0102

⇒ fwrite, freac

⇒ fwrite

This library function writes data from the array pointed to, by ptr to the given stream.

Size-t fwrite (const void *ptr, size-t size, size-t nmembr,

array

size of each array

Store it →

FILE *stream)

ptr : pointer to array of elements to be written.

size : size in bytes of each element to be written.

nmembr : No. of elements, each one of size of size bytes.

Stream : pointer to a FILE object that specifies an output stream.

size-t : unsigned int

function returns total no. of elements successfully returned as size-t object

otherwise it will return error.

```
#include <stdio.h>
int main(){
    FILE *fp;
    char str[] = "this is AAC";
    fp = fopen ("file.txt", "w");
    fwrite (str, 1, sizeof(str), fp);
    fclose(fp);
    return 0;
}
```

Phone: +91 844 844 0102

fread : This C library function reads data from the given stream into the array pointed to by ptr.

Size-t fread(void *ptr, size-t size, size-t nmemb,
FILE *stream)

Can point to
any data type

ptr : pointer to block of memory with a min size of size * nmemb bytes.

size : size in bytes of each element to be read.

nmemb : This is number of elements, each one with a size of size bytes.

Stream : This is pointer to a FILE object specifies an input stream.

Block → array

fseek : This C library function sets the file position of the stream to the given offset.

int fseek(FILE *stream, long int offset, int whence)

stream : pointer to file object that identifies stream.

offset - No. of bytes to offset from whence.

whence : position from where offset is added. specified

Mail: gatcse@appliedroots.com

SEEK_SET : Beginning of file

SEEK_CUR : Current position of file pointer

SEEK_END : End of file

funcn returns 0 if successful

Non-zero else

#WAP for usage of fseek() function

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
FILE *fp;
```

```
fp=fopen("file.txt", "w+");
```

```
fput ("This is Appliedgate", fp);
```

```
fseek(fp, 7, SEEK_SET);
```

want to move 7 bytes ahead

```
fput ("C programming language", fp);
```

```
fclose(fp);
```

```
return 0;
```

```
}
```

O/P: This is C programming language

fseek helps to perform random access

Mail: gafecse@appliedroots.com

Phone: +91 844-844-0102

fTell

Returns the current file position of the given stream.

`long int ftell(FILE *stream)`

abcd^{efghi}j^{klm}n^{opq} — returns 10

Input:

rewind:

Sets the position of file to the beginning of the file of the given stream.

`void rewind(FILE *stream)`

Ex:

abcd^{efghi}j^{klm}n^{opq} → after rewind it will ~~start~~ start pointing to start.

⇒ Moving the control with in file.

Remove

Deletes the given filename so that it is no longer accessible.

`int remove(const char *filename)`

Mail: gatcse@appliedroots.com

fflush()

- This C library function flushes the output buffer of a stream.

`int fflush(FILE *stream)`

Stream = pointer to a file object that specifies a buffered stream.

Returns : returns 0 if success
— If error occurs, EOF is returned.
error indicator is set (i.e. feof)

[15.1] Enumerators : enum keyword in C .

\Rightarrow just like struct, union, enum is also a user-defined datatype.

Month Jan, Feb, March -- .

(+) return month { Jan, Feb, Mar, Apr, May } ; // Defining
| |
keyword identifier L Identifiers

enum month {m1, m2};
m1 = JAN } Declaring a var
m1 = JUN

(2) return month of Jan, feb = 4, March , Abr, May = 10 %;

⑤ `Printf("Y.S", m);` X internally m is integer & here I am treating it as string

④ enum set1 { abc, def, ghi } ;
enum set2 { abc, xyz, mnop }
int def = 0; X

Whatever you are
using in one
enum can not be
used in the same
scope.

⇒ internally helps to increase code readability.

[15.2] Function Pointers Phone: +91 841-844-0102 (Pointers to functions)

⇒ code is also placed in the memory

each function) → place/location in memory
 ↓
address

```
int f()
{
}
```

```
int main()
```

```
{
}
```

```
printf("%d", f);
```

```
}
```

fn name

address of the actual fu

Pointer to a function :

```
float func (int, int); // declaration of func
```

```
float (*fp) (int, int); // fp is a pointer to a function
                        that takes 2 int argument
                        & returns a float
```

fp = func;

address of function

①

```
float a = func (10, 20);
```

②

```
float a = (*fp) (10, 20);
```

Pass a function address as an argument to another fn.

```
void ft (ptr);
```

```
void ft (int, void (*fp)(int));
```

and arg is fun pointers
 Pointing to a func with
 one int arg as ret

Mail: gatose@appliedroots.com

Phone: +91 844-844-0102

main ()

~~Applied Roots~~

f (lo, f1);

}

↳ address of function f1

void f1 (int a)

{
}

void f (int x, void (*
f1 (int));

{
}

array to function pointer

float (*fp[4]) (float, float);

fp is an array of function pointers where each func has return type float & 2 float argument.

float add (float, float);
 float sub (float, float);
 float mul (float, float);
 float div (float, float);

fp[0] = add;
 fp[1] = sub;
 fp[2] = mul;
 fp[3] = div;

float (*fp[4]) (float, float) = {add, sub, mul, div};

Declaration

assignment

① int *f (void) → fn that returns int * & takes no args

② int (*fp) (void) → fn pointer

③ int (*fp[4]) (void) → array of fn pointers
where each fn has no arg

Mail: gatecse@appliedroots.com

④ $\text{int } *(\ast \text{fp})(\text{void}) \rightarrow$ fn pointer to a fn that
returns an integer for

⑤ $\text{int } *(*\text{fp}[4])(\text{int } a, \text{int } b)$

\uparrow

return type

$\underbrace{\hspace{100px}}$

array of
fn pointers