Now we know the default gateway IP address but in LAN communication we should require mac address by using ARP, we can resolve it.

* Now Host A will be creating two headers L2 header & L3 header.

L2 header

Source Mac	Destination mac
A	R_1

L3 header

Source IP address	Destination IP address
Host A	Host B

* Now it will go to Swith. Switch will be checking in mac table destination mac address which port it is connected that port will be forwarded Now packet will move from R_1 to R_2. Now R_1 will remove L2 header.

L3 header

Source IP address	Destination IP address
Host A	Host B

* Every router having the routing table. Now R_1 will be checking the routing table destination table B is connected or not. Now it's not connected.

* Now R_1 again fetching the default gateway IP address. The default gateway IP address of R_1 is R_2. Now we know the IP address of R_2 But in Lan communication we should require mac address then only we can communicate By using ARP we can resolve it. Now R_1

* Now R_1 Creating the L3 header & L2 header.

L2 header

Source Mac	Destination MAC
R_1	R_2

* Now Packet will move from R_1 to R_2. Now R_2 will remove L2 header R_2 will be checking in routing table destination B will be connected as not Here goes it is connected.

* Now It knows the destination IP address of B but in LAN communication we should require mac address by using ARP we can resolve it.

* Now will will go to the switch, switch will be checking in the mac table destination mac address which port will be connected that port it will forward

Eg:-

L-ads-5665 :~> ping www.google.com

PING www.google.com (172.217.12.100) 56 (84) bytes of data.

bytes from el-in-f4.1e100.net (172.217.12.100) : icp-seq=1 ttl=53 time=204 ms

bytes from

--- www.google.com ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 1001ms

t min/avg/max/mdev = 204.058/204.665/205.336/0.568 ms

L-ads-5665 :~>

OSI Model :- [open system interconnection] Set of rules - Protocol.

- Application layer — DATA
- Presentation layer — DATA
- Session layer — DATA
- Transport layer — Segments
- Network layer — packets
- Data layer
- Physical layer

(Application layer, Presentation layer, Session layer, Transport layer — Software layers)
(Network layer, Data layer, Physical layer — Hardware layers)

Sender

Eg :- IP address
A — 0-127
B → 128-191
C → 192-223
D → 224-239
E → 240-255

class D is multicast address
class we have to use special purpose.

Net Id	Host Id
A 1	3
B 2	2
C 3	1
D	

Subnet Id :- large broadcast domain split into multiple small broadcast domain.

1) How many subnets is there in subnet mask

A) 192 — $2^2 = 4$ subnets
11000000
128 64 32 16 8 4 2 1
8 4 2 1

2) How many hosts will be connected?

0000001.00001001.00000010
8 4 2 1
1 0 0 1

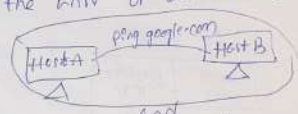
$2^{6off} - 2 = 2^6 - 2 = 62$ hosts will be connected

2

PING :- I want to ping www.cam google.com

* host A will interact with the DNS [Domain name system] it resolves domain name into IP address.
* first it will check wheather the destination IP address is connected within the LAN or outside the LAN.

How it will check?



A) Based on the subnet mask
If the subnet mask is same for sender and receiver then it is within the LAN. If not means outside the LAN.

Assumption in case within the LAN.

* Host A want to communicate with Host B. Host A know the destination IP address of Host B. But in LAN communication we should require mac address then only we can communicate. Every host will be having ARP catching table. Host A will be checking in the ARP catching table. If the mapping. Incase it is not there means by using path we can resolve it

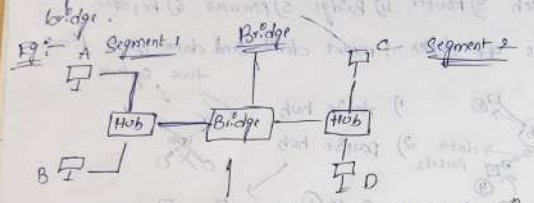
) How it connecte outside the LAN?

A) Now host A will be fetching the default gateway IP address. The default gateway address of host A is ARP.

Note :- whenever we assign the IP address along with IP address. It will assign some configuration parameters such as subnet mask Id default gateway IP address, DNS server and least time.

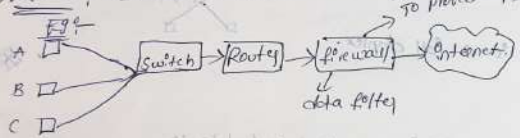
4) Bridge :- It connects two different segments is called bridge.

Eg :-



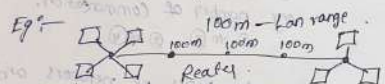
* Bridge is stored all devices mac address by using tabular form.

5) Firewall :- It allows secure data only.



6) Repeater :- By using Repeaters it stops can networks.

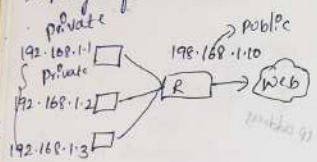
Eg :-



Internet protocol → set of rules

IP address :- heart of networking. Ex :- home base.

* Every systems has different Ip address. Ex :- face of humans



Types of Ip address we recognise.
public
private
dynamic - Temporary
static - permanent.

IPv4 (version 4) :- Numeric address (32 bits)

four octates.

172 . 16 . 254 . 1

max range 255

8 bits

8x4 = 32 bits

Decimal to binary :-

172 . 16 . 254 . 1

32 bits

10101100 - 172

00001100 - 16

11111110 - 254

00000001 - 1

128 64 32 16 8 4 2 1

240

Class	127 - loop address octate range	format	Network ID : Host ID
Class A	1 - 126	1/4	3/4
Class B	128 - 191	2/4	2/4
Class C	192 - 223	3/4	1/4
Class D	224 - 239		
Class E	240 - 255		

172 . 16 . 254 . 1 ← class B.

network address (ID) | host addresses (ID)

173 . 16 . 254 . 1 class A

network | Host ID

Ipv6 (version 6) :- 128 bits

16 bits

Eg :- 2001 : 0BD1 : AC20 : FE01 : 0000 : 0000 : 0000 : 0000

Hexadecimal format → divide : format

* Every hexadecimal have 4 bits.

2^{128}

0 - 9
A - F
A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

2001 :

8 4 2 1

0010 0000 0000 0001

0BD1 :

0000 1011 1101 0001

Eg :- FDE0 : 0074 : 0000 : 0000 : B0FF : 0000 : FFF0 : 0028

FDE0 : 74 : 0 : 0 : B0FF : 0 : FFF0 : 28

FDE0 : 74 : : B0FF : 0 : FFF0 : 28 ← don't use two times in one IP address.

MAC Address :- [Media access control] 48 bits.

6 x 8 = 48

Eg :- 00 : 1A : 3F : F1 : 4C : C6

1 2 3 4 5 6

* mac address is assigned to NIC (network internet card)

00000000 : 00011010 : 00111111 : 11110001 : 01001100 : 01000110

128 64 32 16 8 4 2 1

wireless wire

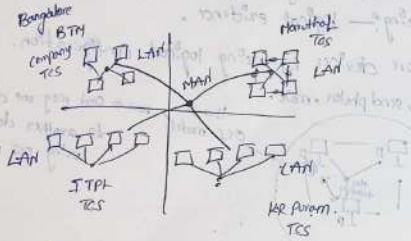
A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

8 4 2 1

WAN (wide area network) :- wireless more than two devices connect by using internet is called WAN. (public)

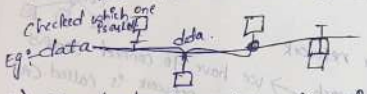
Ex:- zoom, teams, what's up group,

MAN :- To connect more than two LANs is called MAN.

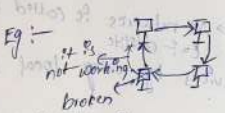


*** Topologies :- (sub topic of LAN)

1) Bus topology :- Slow process, Small count of systems we have that time we use bus topology.



2) Ring topology :- data flow is circle format once one system is cracked next data is not connected to next system

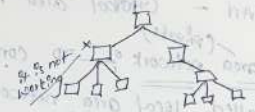


3) Star topology :-

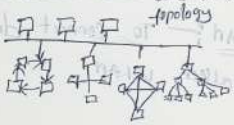
4) Mesh topology :-

Disadvantage we require more wires

5) Tree topology :-



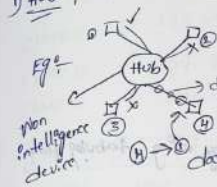
6) Hybrid topology :- All are used in the topology



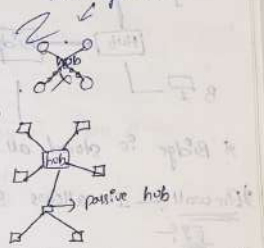
Networking devices :-

1) Hub 2) Switch 3) Router 4) Bridge 5) Firewall 6) Repeater

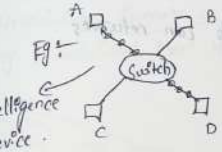
1) Hub :- It is off duplex → means don't send data at a time



1) Active hub
2) passive hub



2) Switch :- It is a full duplex.



direct sends data to
4 to 1 at a time we send any number of communication.
1 to 3 2 to 4

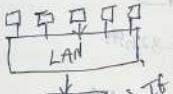
3) Router :- It connects two different types of networks and also internet.

192.162.10.1
192.162.11.2
} IP address

Ip address :-

1) dynamic Ip address (Private) :- we assign ip address
2) static Ip address (Public) :- (unique address)

different assign IP address anyone



→ It connects same type of networks.

Eg:- same 192.162.10.1 / 192.162.10.2

→ It connects two different types of networks → diff 192.162.11.1 / 192.162.10.2

(or) to connect internet also.

Digital marketing :— It's a fantastic way to grow business locally, globally using digital media channels.

1) Think 2) write 3) Prioritise 4) start.

Jobs
Trained, All power

Networking

* Networking means to connect two or more devices is called networking. They are 2 types.

1) Hardware networking :— Physical existence
* Connect two or more devices by using wire (or) connector (or) any thing.

2) Software networking :— logical existence.
* Connect two or more devices by using logical configuration.
Ex :- wifi, msg, send photos, mails.

Eg :-

Theory OSI models — how to send msg one device to another device by using OSI model.

Networking Basics :— 1960s - 1970

1) LAN – local area network.
2) WAN – wireless local area network
3) WAN – wide area network
4) MAN – Metropolitan Area network
5) CAN – Campus Controller area network → we have to connect campus area network is called CAN
6) SAN – Storage area network → memory, storage Access
7) PAN – Personal area network
8) GAN – Global area network → All networks is called Global.
Ex :- satellite

Local area network (private) :— To connect devices by using local area is called local area network.

Eg :- office.

local area only Restaurant wifi concept.

WLAN :— To connect devices by using wireless local area is called WLAN.

without using wire.

OSPF (open Shortest path first) :—

L_2 – frames
L_3 – Packets

10·40·50·0 ⟶ Subnet Id . Broadcast address
10·40·50·64 ⟶ 10·40·50·63
10·40·50·128 ⟶ 10·40·50·127
10·40·50·192 ⟶ 10·40·50·191
⟶ 10·40·50·255

256 − 192 = 64

1st subnet id – 2nd subnet id.

255·255·255·128
·224

128 64 32 8 4 2 1
1 0 0 0 0 0 0 0

on $2 \Rightarrow 2^1 = 2$ subnets

of $2 \to 2^7 = 128 - 2$
= 126 Host

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
+64+32+16+8+4+2+0

256 − 128 = 128

Id 10·40·50·0 ⟶ 10·40·50·127
~~10·40·50·64 ⟶ 10·40·50·127~~
10·40·50·128 ⟶ 10·40·50·255

128 + 128 = 256

224

11100000

128 + 64 + 32 = 224

224

on $2 \to 2^3 = 8$ subnets
2×2×2 = 8

of $2 \to 2^5 = 32 - 2$
= 30 Host

256 − 224 = 32

Id			
10·40·50·0 – 10·40·50·31	10·40·50·160 – 10·40·50·191		
10·40·50·32 – 10·40·50·63	10·40·50·192 – 10·40·50·223		
10·40·50·64 – 10·40·50·95	10·40·50·224 – 10·40·50·254		
10·40·50·96 – 10·40·50·127			
10·40·50·128 – 10·40·50·159			

Areas :—
1) Stub area $[T_1, T_2, T_3]$
2) Not so stubby area $[T_1, T_2, T_3, T_7]$
3) Total stub area $[T_1, T_2]$
4) Not so total stub area $[T_1, T_2]$

Area 0 → Back bone area

```
int main()
{
 int status;
 pthread_t t1, t2;
 pthread_create(&t1, NULL, odd, NULL);
 pthread_create(t2, NULL, even, NULL);
 pthread_join(t1, NULL);
 pthread_join(t2, NULL);
 return 0;
}
```

gcc sample.c –lpthread — compile time

o/p:- odd : = 73
even : = 78

Pthread_join :— waiting for the thread termination.

*** Diff b/w thread & process :—

thread	process
* If we are creating new thread the operating system does not allocate separate memory it will sharing the memory for the correct process.	* If we are creating new process the operating system allocated separe memory for the new process
* It will share stack segment	* It will share stack segment
* They is no need of separe IPC.	* when process want to communicate should be required any one of the IPC (Inter process Communication)

Zombies.c :—

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
  int pid;
  int n;
  char *p;
  pid = fork();
  if (pid == 0)
  {
    n = 4;
    p = "child";
  }
  else
  {
    n = 10;
    p = "Parent";
  }
  for(n; n>0; n--)
  {
    sleep(5);
    printf ("\n %s \n", p);
  }
}
```

Thread management :—

* Thread is a multiple tasks of execution with in the process is called as a thread.

How to create the thread :—

```
pthread_create(pthread_t * threadid, const pthread_attr *attr,
    void * (*fp) (void *arg);
```

threadid — name of the thread; attr — none / by default passing attributes

fp — function pointer (or) argument

*** Eg :—

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
  int i=0;
  void * odd()
  {
    while (i<100)
    {
      if (i%2==1)
      {
        printf ("\nodd i=%d", i);
        i++;
      }
    }
    void *even()
    {
      while (i<100)
      {
        if (i%2==0)
        {
          printf ("\neven i=%d", i);
          i++;
```

It creates two thread one is odd number one is even number.

zroid.c :-

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
 int pid;
 int n;
 char *p;
 pid = fork();
 if (pid==0)
 {
  n=4;
  p="child";
 }
 else
 {
  wait();
  n=10;
  p="parent";
 }
 for (n;n>0;n--)
 {
  sleep(5);
  printf("\n %s \n",p);
 }
```

how.c :-

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
 fork();
 fork();
 fork();
 printf("\nbangalore");
}
```

orphan.c :- [Parent will be take before the child process child is called orphan process.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
 int pid;
 int n;
 char *p;
 pid = fork();
 if (pid==0)
 {
  n=10;
  p="child";
 }
 else
 {
  n=3;
  p="parent";
 }
 for (n;n>0;n--)
 {
  sleep(5);
  printf("\n %s\n",p);
 }
}
```

int → It will add parent

```
int i=5;
if (--i)
{
 main();
 printf("%d",i);
}
```

st.c :-

```
#include<stdio.h>
void main()
{
```

How to change permissions :—

Chmod ⟶ we can change permissions.

Read a file :—

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
void main()
{
  int fd;
  char buffer[20];
  fd = open ("/home/hp/chandra/c/linux/files/hi.text", O_Ronly);
  if (fd == -1)
  {
    printf ("file is not open !!!!!!!!!!!!!!");
  }
  read (fd, buffer, 3);
  printf ("\n %s", buffer);
  close (fd);
}
```

O/P :— hei

exec.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
  printf ("hi");
  execlp ("ls", "ls", "-lF", NULL);
  printf ("welcome");
}
```

O/P :—

* The exec the correct process replacing with new files.

list of commands

fork.c :— By using the fork we are creating the new process. The operating system allocated separate memory for the new process. By using the V-fork while creating the new process but operating system does not allocate separate memory.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
  int id;
  // printf("\n %d", id);
  id = fork();
  if (id == 0)
  {
    printf ("\nchild");
  }
  else
  {
    printf ("\nparent");
  }
}
```

O/P :— parent child.

* we can't justify which process running 1st file.
* Change should be run first.

pid.c :—

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
  int pid;
  pid = fork();
  if (pid == 0)
  {
    printf ("\nchild = %d", getpid());
    printf ("\nchild parent id = %d", getppid());
  }
  else
  {
    printf ("\nparent = %d", getpid
  }
}
```

O/P :—

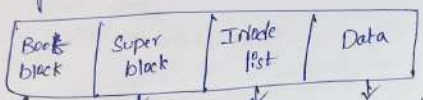
linux

open source.

file management :—

Kernel is a heart of linux.

file system architure :—



It stores os booting files

It stores file system architure.

It stores list of the inode numbers

It stores data only.

file permissions :— 2 types to assign permissions.

1) write → 2
2) read → 4
3) Execution → 1

1) Absolute (numaric)
2) Symbolic (Alphabet).

linux users :—

1) user 2) group 3) all

7 5 4

permissions.

Number	permission type	Symbol
0	No permission	---
1	exe	--x
2	write	-w-
3	exe + wri	-wx
4	Read	r--
5	Read + Exec	r-w
6	Read + write	rw-
7	Read + wri + exe	rwx

How to create a file.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
int inode;
inode = creat("hi.text", 777);
if (inode==-1)
{
printf("\n fail the file Creation\n");
}
else
{
printf("\n file Creation Success\n");
}
}
```

System calls it will be creating the file.

type name permission

o/p :— * Success means it will return inode number.

* fail is -1.

ls → list of files.

Inode number :— It is a positive integer value it will be stores the properties of the file.

ls -i → It shows inode number.

write a file :—

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
void main()
{
int fd;
char buffer[20];
fd = open("/home/hp/chandra/c/linux/file/hi.text", O_RDWR);
if (fd==-1)
{
printf("fail the opening the file");
}
else
{
printf("file successfully open");
}
gets(buffer);
write(fd, buffer, 5);
close(fd);
}
```

string

write & read

data will be writing buffer to fd.

file is a size.

o/p :— file is success it will return fd.

-1 is the failure.

ls -l → check permissions.

ls -lf → check permissions.

get → It read multiple values

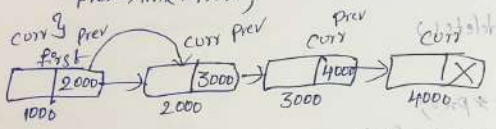
scanf → It read single values.

read write execution

rwx-rwx-r

Eg:- last node deletion

```
void delete()
{
struct node *prev;
curr = first;
while(curr->link != NULL)
{
  prev=curr;
  curr = curr->link;
}
free(curr);
prev->link = NULL;
```



Eg:- Reverse linked list

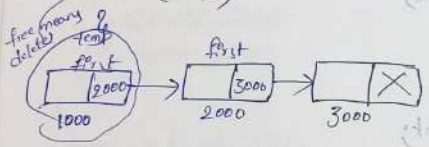
```
void reverse()
{
curr = NULL;
while(first != NULL)
{
  temp = first;
  first = first->link;
  temp->link = curr;
  curr = temp;
}
first = curr;
}
```

Eg:- void display ()

```
{
 curr = first;
 while (curr != NULL)
 {
 Printf ("\n%d", curr->data);
 curr = curr->link;
 }
```

Eg:- Deleting the first node.

```
void delete()
{
 temp = first;
 first = first->link;
 free (temp);
```

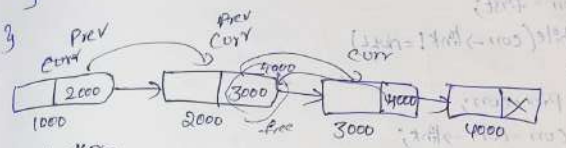


Eg:- Delete the particular position

Eg:- void pos-del (int pos)

```
{
 Struct node *prev;
 int count = 1;
 curr = first;
 while (curr != NULL && pos != count)
 {
  prev = curr;
  curr = curr->link;
  count++;
 }
 if (curr == NULL)
 {
 Printf ("\n there is no sufficient nodes");
```

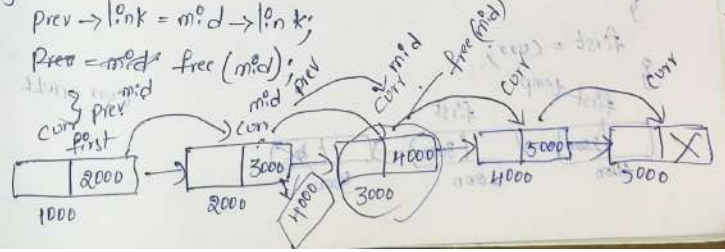
```
else
{
 prev->link = curr->link;
  free (curr);
}
}
```



count = 3
pos = 3

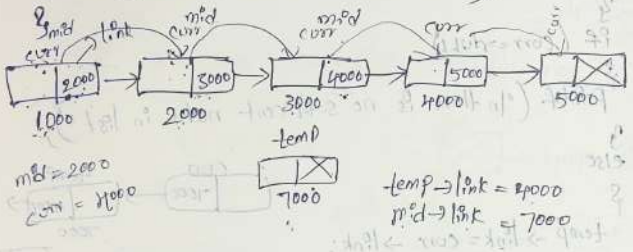
Eg:- middle node deletion

```
void mid_delete()
{
 Struct node *prev;
 struct node *mid = first;
 curr = first;
 while (curr->link != NULL)
 {
  curr = curr->link;
  if (curr->link != NULL)
  {
   prev = mid;
   mid = mid->link;
   curr = curr->link;
  }
 }
 prev->link = mid->link;
 Free = mid free (mid);
```



middle adding a node :—

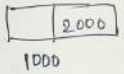
```
void mid_add (int data)
{
 temp = (struct node*) malloc (sizeof(struct node));
 temp->data = data;
 temp->link = NULL;
 Struct node *mid = first;
 curr = first;
 while (curr->link != NULL)
 {
  curr = curr->link;
  if (curr->link != NULL)
  {
   mid = mid->link;
   curr = curr->link;
  }
 }
 temp->link = mid->link;
 mid->link = temp;
 printf ("\n mid->data = %d", mid->data);
```



last adding a node :—

Eg:—

```
void last_pos (int pos, int data)
{
 temp = (struct node*) malloc (sizeof (struct node));
 temp->data = data;
 temp->link = NULL;
 int count = 1;
 curr = first;
 Struct node *p = first;
 while (curr->link != NULL)
 {
  if (pos != count)
  {
   curr = curr->link;
   count++;
  }
  else
  {
   curr = curr->link;
   p = p->link;
  }
 }
 temp->link = p->link;
 p->link = temp;
}
```



Linked list :—

Creation

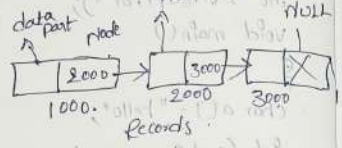
Eg :—
```
#include <stdio.h>
struct node
{
  int data;
  struct node *link;
};
struct node *first = NULL;
struct node *temp = NULL;
struct node *curr = NULL;
void creat (int);
void display (void);
void main()
{
  creat (10);
  creat (20);
  creat (30);
  display ();
}
void creat (int data)
{
temp = (struct node *) malloc (sizeof (struct node));
temp -> data = data;
temp -> link = NULL;
if (first == NULL)
{
   first = temp;
}
else
{
curr = first;
while (curr -> link != NULL)
{
curr = curr -> link;
}
curr -> link = temp;
}
}
```

Syntax

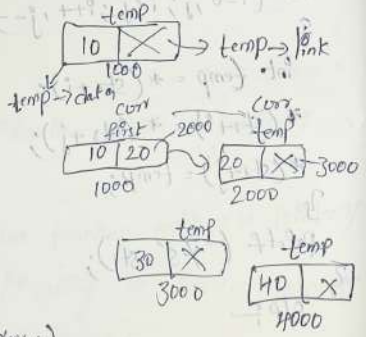
data part

eg :— student information.

int number;
char name;

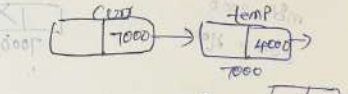


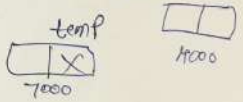
dynamic memory allocation

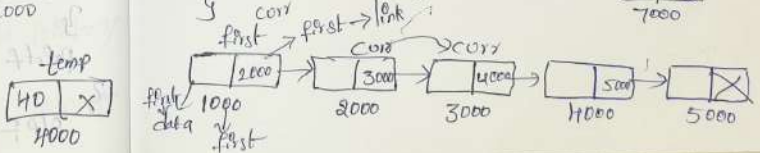


Adding a node in particular position :—

Eg :—
```
void pos_add (int pos, int data)
{
temp = (struct node *) malloc (sizeof (struct node));
temp -> data = data;
temp -> link = NULL;
int count = 1;
curr = first;
while (curr != NULL && pos != count)
{
  curr = curr -> link;
  count ++;
}
if (curr == NULL)
{
printf ("In there is no sufficent nodes in list");
}
else
{
  temp -> link = curr -> link;
  curr -> link = temp;
}
```







* a is a pointer it will only stores the address of pointer arrays.

Eg:- int (*p)(int, int); → function pointer we use in call back funtion.

* p is a fonction pointer it will be passing integer argument and integer argument; return the integer.

Eg:- Declare the fonction pointer it will be passing the character, pointer and integer pointer return the character pointer.

~~char(*p)(char*, int*)~~ ~~int char~~

Eg:- char *(*p)(char*, int*)

* size is 4 bytes.

Eg:- int (*p[3])(int, int); → array of function parter.

* size is 12 bytes.

* Function pointer :-

To add two variable by using call back fonction.

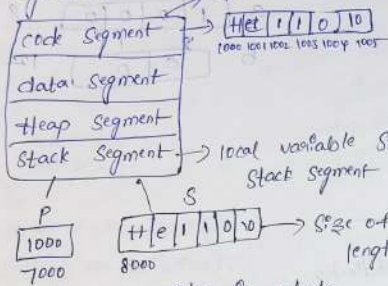
Eg:- #include <stdio.h>
int Sum (int, int);
void main()
{
int (*p)(int, int); /* fonction pointer declaration*/
int a=10;
int b=20;
p=Sum; /* assaing the function address to pointer*/
int result = p(a,b); /* calling the fonction pointer*/
Printf ("\n result = %d", result);
}
int Sum (int a, int b)
{
int c=a+b;
return(c);
}
o/p :- result = 30

Eg:- String reverse using the fonction pointer.

```
#include <stdio.h>
int reverse(char *);
void main()
{
char a[] = "hello";
int (*p)(char *); /* fonction pointer declaration*/
p= reverse; /* assigning function address to pointer*/
p(a); /* calling the function pointer*/
}
int reverse (char *str)
{
int i=0;
int j= Strlen(str)-1;
for(i=0, j; i<j; i++, j--)
{
int temp = *(str+i);
*(str+i) = *(str+j);
*(str+j) = temp;
}
Printf ("%s", str);
}
```

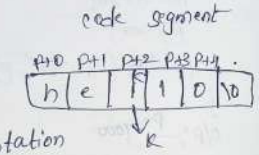
o/p :-

Memory Segments :— It is a readable section.



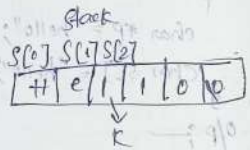
* any address of pointer is 4 bytes.

Eg :— #include <stdio.h>
void main()
{
char *p = "hello";
char S[] = "hello";
*(p+2) = 'k'; ⟶ Segmentation error.
printf ("%s", p);
}
O/P :— Runtime error



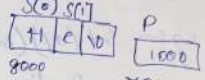
* Code Segment is only readable but we can try to write. So that it gets Segmentation error.

Eg :— void main()
{
char *p = "hello";
char S[] = "hello";
S[2] = 'k';
printf ("%s", S);
}
O/P :— Hek lo



* Stack Segment is writeable section.

Eg :— main()
{
char *p = "hello";
char S[] = "he";
p = S;
printf ("%s", p);
}
O/P :— He



* pointer is not a constant but value is a constant.

Eg :— S = p
Pf ("%s", S);
O/P :— Error

* value is not a constant but pointer is constant.

* Array starting address we can't change.

Eg :— #include <stdio.h>
void main()
{
const char *p = "hello"; ⟶ value is constant !
char *const q = "hello"; ⟶ pointer is a constant
const char *const r = "hello"; ⟶ Both will be constant
char const *p = "hello"; ⟶ value is a constant.
}
O/P :—

Complicated declarations :—

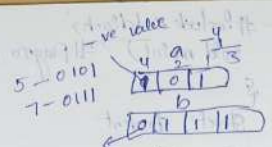
* char *a[3]; ⟶ array of pointer.
* address of character is 12 bytes.

Eg :— char x, y, z;
a[0] = &x; = 1000
a[1] = &y; = 1001
a[2] = &z; 1002.

Eg :— char (*a)[3]; ⟶ pointer to an array.
* a size is 4 bytes.

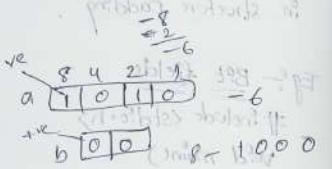
Eg:-
```
#include <stdio.h>
void main()
{
 struct xyz
 {
  int a:3;
  int b:4;
 };
 struct xyz s={5,7};
 printf ("a=%d", s.a);
 printf ("b=%d", s.b);
}
```

o/p :- a = -3
b = 7



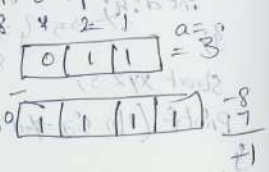
Eg :-
```
#include <stdio.h>
void main()
{
 struct xyz
 {
  int a:4;
  int b:2;
 };
 struct xyz s={10,8};
 printf ("a=%d", s.a);
 printf ("b=%d", s.b);
}
```

o/p :- a = -6
b = 0

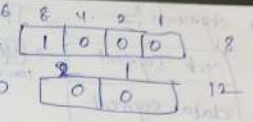


Eg :-
```
 int a:3;
 int b:4;
};
struct xyz s={11,15};
```

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



Eg :- a : 4
b : 2



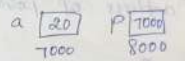
S = {8, 12}

a = -8
b = 0

Pointers :-

Memory allocation :-

Eg :-
```
#include <stdio.h>
void main()
{
 int *p;
 int a=20;
 p=&a;
 printf ("%d", p);
}
```



o/p :- p = 7000
*p = 20
++p = 7004
p++ = 7000
*++p = 7004 value
++*p = 21

Eg :-
```
#include <stdio.h>
void main()
{
 char *p = "hello";   --> character pointer
 char s[] = "hello";  --> string
}
```

o/p :-

length of the character is 5

Eg:- #include <stdio.h>
void main() → #Pragma pack(1) — (2) — invalid
{ (2)
Struct student size=6
{
int a;
char b;
};
Struct student s1;
Printf ("\n size=%d", sizeof(s1));
}

o/p:- size = 5

Structure padding :-
* memory will be waste in structure padding.

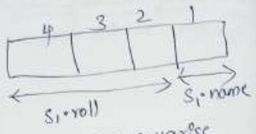
Pragma pack
* Program execution speed is less.

Eg:- Bit fields
#include <stdio.h>
void main()
{
Struct xyz
{
int b:8; → 8 bits
int a:10; } 32 bits
int c:10; } 4bytes
int d:4; → 5th } 33 bits
8 bytes — size = 8
};
Struct xyz s;
Printf ("\n size=%d", sizeof(s));
}
o/p:- size = 4

Unions :— Collection of dissimilar (different) datatype.

Eg :—
```
#include <stdio.h>
void main()
{
    union student
    {
        int roll;      } size 4bytes
        char name;
    };
    union student s1;
    scanf("%d", &s1.roll);
    scanf("%c", &s1.name);
    pf("roll = %d\n", s1.roll);
    pf("name = %c\n" s1.name);
}
```
Declaration of variables: {s1, s2;}

* we consider here highest data type size.



it will collarise override with binary form value of name.

o/p :— 10 a
roll = 97
name = a

512a
roll = 609
name = a

10a
roll = 97 → a - ASCII value 97.

Structure	Union
* Size of the structure is Sum of all the variable size.	* Size of union is highest variable size.
* In Structure all the variable are valid.	* Only one variable is valid.

Syntax :— (Declaration)
```
struct Tagname
{
    datatype ele1;
    datatype ele2;
      ,,       ,,
};
```

Syntax :—
```
union tagname
{
    datatype ele1;
    datatype ele2;
      ,,       ,,
};
```

Eg :—
```
struct employee      → S [28bytes]
{
    int eno;           4
    char ename[20];    20
    float salary;      4
}
```
28 bytes of memory allocated

Eg :—
```
union employee
{
    int emp;          → 4
    char ename[20];   → 20
    float salary;     → 4
};
```
[20bytes] of memory allocated

* The most difference Structure and union is memory allocation.

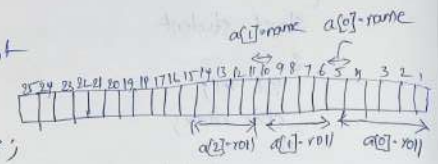
Declaration of a union variable :—

Eg :—
```
main()
{
    union employee e1, e2, e3;
}
```

Array of Structures :—

```
Eg :— #include <stdio.h>
       void main()
       {
         struct student
         {
           int roll;
           char name;
         };
         struct student a[5];
         int i;
         for (i=0; i<5; i++)
         {
           scanf ("%d", &a[i].roll);
           scanf ("%c", &a[i].name);
         }
         for (i=0; i<5; i++)
         {
           printf ("roll=%d\n", a[i].roll);
           printf ("name=%c\n", a[i].name);
         }
       }
```

5*5 = 25 bytes



o/p :—

* Eg:-
```
#include <stdio.h>
void main()
{
  char *P;
  P=(char*)malloc(10);
  strcpy(P,"hello");
  free(P);
  P=NULL;
  printf("%s",P);
}
```

Structures:- It is a collection of similar and dis-similar data types. * array stores only similar data type but structure stores all data types.

Eg:-
```
#include <stdio.h>
void main()
{
```

Eg:- array
```
int a[5];
```

Eg:- structures
```
int roll;
char name;
float marks;
```

Syntax:- struct identity/tagname

Keyword: struct student


```
{
  int roll;
  char name;
};
```
→ every structure defination end with ;

5 bytes data type

```
struct student S1;
scanf("%d",&S1.roll);
scanf("%c",&S1.name);
printf("roll=%d\n",S1.roll);
printf("name=%c\n",S1.name);
}
```

o/p:- 10 a
roll = 10
name = a

%d → It will print a ASCII value 97

Structure follows four rules

1) Define a structure.
2) Declare structure variable
3) initialize structure members/elements
4) Access the structure members

```
struct student S1={10,"Rekha"};
```

* member access operator (.) dot.

S1.name, S1.roll

```
struct student S1,S2,S3;
```

we can create variables like this also.

copy of one structure to another structure:-

Eg:-
```
#include <stdio.h>
void main()
{
  struct student
  {
    int roll;
    char name;
  };
  struct student S1,S2;
  scanf("%d",&S1.roll);
  scanf("%c",&S2.name);
  S2.roll=S1.roll;
  S2.name=S1.name;
  Pf("roll=%d\n",S2.roll);
  pf("name=%c\n",S2.name);
}
```

o/p:- 10 a
roll= 10
name = a

Eg:- String copy

```
#include <stdio.h>
void main()
{
  struct student
  {
    int roll;
    char name[10];
  };
  struct student S1,S2;
  S1.roll=10;
  strcpy(S1.name,"chandra");
  S2.roll=S1.roll;
  strcpy(S2.name,S1.name);
  Pf("roll=%d\n",S2.roll);
  Pf("name=%s\n",S2.name);
}
```

o/p:-

2) Calloc :— Syntax :— void * calloc (no. of elements, size of each element);

Eg:—

```
#include <stdio.h>
void main()
{
    int *p;
    p = (int *)calloc(4,5);   // 4 -> number of blocks, 5 -> Each blocks   4x5 = 20
    if (p == NULL)
    {
        printf("\n memory allocation fail\n");
    }
    else
    {
        printf("\n memory allocation Success\n");
    }
}
```

o/p :—

malloc	calloc
* It is single block memory allocation	* It is multiple block memory allocation
* Default value is garbage value.	* Default value is zero.

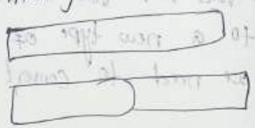
3) Realloc :— By using realloc we can increase and decrease memory allocation.

Eg:—

```
#include <stdio.h>
void main()
{
    char *p;
    p = (char*)malloc(10);
    p = (char*) realloc (p,20);
}
```

o/p :—

* Here we have space. It will allocated the memory.
* we don't have space it will create new memory allocation.



Eg:—

```
#include <stdio.h>
void main()
{
    char *p;
    p = (char*)malloc(10);
    strcpy(p,"hello");
    p = (char*) realloc(p,20);
    printf("\n %s\n",p);
}
```

o/p :— hello

* It will old memory data copy to new memory.

4) Free :— By using free we can reallocate the dynamic memory.

Eg:—

```
#include <stdio.h>
void main()
{
    int *p;
    p = (int *)malloc(10);
    free(p);
}
```

o/p :—

memory leak :— we are allocated memory dynamically but not free the memory is called memory leak.

* find out m.l. By using "valgrind".

Dangling pointer :— we are allocated memory dynamically and free that memory. But the pointer still pointing to that memory allocation it called as dangling pointer.

* functional infact it will be segmentation problem.

```
        a[i] = a[k];
        a[k] = t;
    }
    printf("\n%s", a);
}
```

o/p:-

Highest repeated word :-

```
#include <stdio.h>
void main()
{
char a[] = "abacaaccccba";
int i, j, k;
int flag = 0;
char big;
for (i=0; a[i] != '\0'; i++)
{
  int count = 0;
  for (j = i+1; a[j] != '\0'; j++)
  {
    if (a[i] == a[j])
    {
      count++;
      for (k = j; a[k] != '\0'; k++)
      {
        a[k] = a[k+1];
      }
      j--;
    }
  }
  if (flag < count)
  {
    flag = count;
    big = a[i];
  }
}
```

```
printf("charter = %c flags
        = %d", big, flag);
}
```

o/p:-

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]
a	b	a	c	a	a	c	c	c	c	c	b	a
a	b	c	a	a	c	c	c	c	c	b	a	\0

count = 1
a[i+1] = c
flag = 1
big = 1

Dynamic memory allocation :- functions are four types.
* These functions are in <stdio.h>

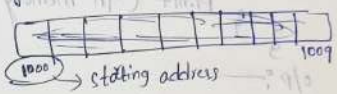
Static memory :-
* Compile time allocated the memory.
* It is a fixed memory.
Eg :- int arr[20];

Dynamic memory :- library (or) header file.
* Run time allocated the memory.
* It is fixed memory in runtime.

1) malloc :- By using the malloc allocated the memory.
Syntax :- void * malloc (size)
* pointer stores address of memory.

Eg :-
```
#include <stdio.h>
void main()
{
  int *p;
  p = (int *) malloc(10);
  if (p == NULL)
  {
    printf("\n memory allocation fail\n");
  }
  else
  {
    printf("\n memory allocation Success\n%d", p);
  }
}
```

type casting



o/p:-
* malloc is success it will returns starting address.
* malloc is fail it will returns NULL.

Type casting :- one data type it will assign to another data type by using the type casting.
* The memory allocation by default (void *) data type. → void points
* malloc default value is Garbage value.
* void * :- void * is a generic pointer it will be pointed to a new type of the data, by using the type casting we need to convert.

Substring :—

```
#include <stdio.h>
#include <string.h>
void main()
{
 char a[] = "chxyzb";
 char b[] = "xyz";
 int i, j=0, flag=0, count=0;
 int match = strlen(b);
 for (i=0; a[i]!='\0'; i++)
 {
  if (b[j]!='\0')
  {
   if (a[i]==b[j])
   {
    j++;
    count++;
    if (match==count)
    {
     flag=1;
     break;
    }
   }
   else
   {
    if (count)
    {
     j=0;
     count=0;
     i--;
    }
    j=0
    count=0;
   }
 }
}
```

* count white spaces
* null character is not counted

match = 3

a	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
	c	h	x	y	z	b	\0

b	b[0]	b[1]	b[2]	b[3]
	x	y	z	\0

```
 if (flag)
 {
  printf ("substring found");
 }
 else
 {
  printf ("substring not found");
 }
}
```

o/p :- substring found.

word Swapping :—

```
#include <stdio.h>
void main()
{
 char a[] = "how are you";
 int i, j, k;
 i=0;
 for (j=0; a[j]!='\0'; j++)
 {
  if (a[j]==' ')
  {
   for (k=j-1, i; i<k; i++, k--)
   {
    char t = a[i];
    a[i] = a[k];
    a[k] = t;
   }
   i=j+1;
  }
 }
 for (k=j-i; i; i<k; i++, k--)
 {
  char t = a[i];
```

i j

a[0]	(1)	[2]	k [3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
h	o	w		a	r	e		y	o	u	\0

w	o	h		a	r	e		y	o	u	

i j k

t = h

i = w

k = h

t = a

i = e

k = a

w	o	h		e	r	a		y	o	u	\0

Using gets() & puts() function in C:-

gets() :- It doesnot take space separate characters.

Eg :-
```
#include<stdio.h>
int main()
{
char name[50];
pf("\n enter your name:");
gets(name);
pf("\n your name is:");
puts(name);
}
```

O/P :- 1) enter your name : Rekha
Rekha
your name is : Rekha

2) enter your name :
Telugo scit tutorials
your name is :
Telugo scit tutorials

Eg :-
```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[50];
clrscr();
gets(name); pf("\n enter your name:");
//sf("%s",name);
pf("\n your name is :%s", name);
puts(name); getch();
}
```

O/P :- 1) enter your name : Rekha
your name : Rekha

2) enter your name : Sure kha
your name is : Sure

Strings read -%s
scanf("%s",str);
#° hello how
space separate characters
it not takes.
scanf("%[^\n]s",

puts() :- It takes space separate characters.

sf("%[^\n]s", name);

O/P :- enter your name:
Sure kha
your name is : Sure kha

String handling functions :-

Eg :- Swapping string

```
#include<stdio.h>
void main()
{
char a[] = "how are you";
int i,j,k;
j=10
for(i=0,j=strlen(a)-1; i<j; i++,j--)
{
char t = a[i];
a[i] = a[j];
a[j] = t;
}
i=0;
for(j=0; a[j]!='\0'; j++)
{
if(a[j]==' ')
{
for(k=j-1, j; i<k; i++, k--)
{
char t=a[i];
a[i] = a[k];
a[k] = t;
}
i=j+1;
}
}
for(k=j-1, i; i<k; i++, k--)
{
char t = a[i];
a[i] = a[k];
}
a[k] = t;
Printf("\n%s", a);
```

a[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
h	o	w		a	r	e		y	o	u	\0

u	o	y		e	r	a		w	o	h	\0

y	o	u		a	r	e		h	o	w	\0

O/P :-
you are how

```
#include <stdio.h>
void main()
{
char a[] = "abaacb";
int i,j,k;
for(i=0; a[i]!='\0'; i++)
{
for(j=i+1; a[j]!='\0'; j++)
{
if(a[i]==a[j])
{
for(k=j; a[k]!='\0'; k++)
{
a[k]=a[k+1];
}
j--;
}
}
}
printf("%s",a);
}
```

o/p :- abc

String duplicate :-

Eg :-
```
#include <stdio.h>
void main()
{
char a[] = "abaacbbaccb";
int i,j,k;
for(i=0; a[i]!='\0'; i++)
{
int count=0;
for(j=i+1; a[j]!='\0'; j++)
{
if(a[i]==a[j])
{
count++;
for(k=j; a[k]!='\0'; k++)
{
a[k]=a[k+1];
}
j--;
}
}
printf("%d%c", count, a[i]);
}
```

o/p :- 3a3b2c

my part

Just check

Declaration & initialization of strings :-

* A string is an array of characters.

1) By using char Array

Syn :- char strname[size] = {'H','i','!'};

* Characters may be → alphabet/digit/symbol. special

Eg :- char name[5] = {'s','u','r','i','\0'}; → Null character

* size is optional

Eg :- char name[] = {'s','u','r','i','\0'};

2) By using string literal

Syn :- char strname[size] = "strvalue";

Eg :- char name[5] = "suri";

* We should use double quotes :-

* Here we are not taken null character at end of the string. By default system automatically appends at end of the string null character.

* char strname[] = "strvalue";

Eg :- char name[] = "SCIT";

String :- String is nothing but an array of characters. i.e char Array is called a string. Eg:- char name[5];

Array → int Arrayname[size];
* String → char Stringname[size];

```
#include <stdio.h>
void main()
{
char a[] = {'h','e','l','l','o'};
char p[] = "hello\0hi";
}
```

here it take automatically null character.

hello\0hi → here ends null character → o/p:- hello

o/p:- 9

count number of characters (or) length of the string.

Eg:-
```
#include <stdio.h>
void main()
{
char a[] = "hello";
int count=0;
int i;
for (i=0; a[i] != '\0'; i++)
{
count++;
}
printf("%d", count);
}
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
H	e	l	l	o	\0

char a[5]; → 0 1 2 3 4 (1001 1002 1003 1004 1005)

a[0] != No
a[1] =

O/P:- 5

* The character array the last element always as the null character. | h | e | l | l | o | \0 |

* String length we cannot count null value.
* String size we count null value.

Eg:-
```
#include <stdio.h>
void main()
{
char a[] = "hello";
char p[10];
int i;
for (i=0; a[i] != '\0'; i++)
{
p[i] = a[i];
}
p[i] = '\0';
printf("\n a = %s", p);
}
```

o/p:- hello

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
h	e	l	l	o	\0

a[0] != \0 - T

p[0]	p[1]	p[2]	p[3]	p[4]	p[5]
h	e	l	l	o	\0

321
23-11-2022
R. Surekha

Eg:-
```
#include <stdio.h>
void main()
{
char a[] = "hello";
char p[] = "he";
int i;
int flag = 1;
for (i=0; a[i] != '\0' && p[i] != '\0'; i++)
{
if (a[i] != p[i])
{
flag = 0;
break;
}
}
if (a[i] == '\0' && p[i] == '\0' && flag == 1)
{
printf("Equal");
}
else
{
printf("Not equal");
}
}
```

a: | h | e | l | l | o | \0 |
p: | h | e | \0 | | | |

h != h - T
h != h - T
h != h - F
h == \0
h == \0

o/p:-

Eg:-
```
#include <stdio.h>
void main()
{
char a[] = "hello";
int i, j;
for (i=0, j=strlen(a)-1; i<j; i++, j--)
{
char temp = a[i];
a[i] = a[j];
a[j] = temp;
}
printf("%s", a);
}
```

i = 0, 1, 2
j = 4, 3, 2

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
h	e	l	l	o	\0

0 < 4

temp = 'H'
a[i] = 0
temp = e
a[i] = l
a[j] = e

o/p:- olleH

11) Repeat loop
Syntax :- Repeat:
stmts...
END.

12) For Each
Syntax :- for each <customer> :
stmts.....
For each loop END is used to fetch records one by one from one or more tables with or without where clause from the database.
Eg :- For each customer where City = "Boston":
Display city
End.

13) Entry function :-
A) It is used to extract a string from the list. The function takes two parameters one is the position of the string and another one is the list.
Eg :- List = {a, b, c, d}
entry (4, list)
O/P :- d

14) num-Entries :- The no. of entries available in the list.
Eg :- ("This is siratha")
O/P :- 3

15) Types of Blocks :-
Repeat, do, for-each block
Procedure block
Trigger block
Editing block.

16) Trigger block :- It contains one or more statements. Whenever an event occurs the corresponding trigger block will be fired.

17) Editing block :- It is used to get input from the user.
1) Prompt-for 2) Set 3) Update.

18) diff b/w Prompt-for & Set
Prompt-for :- This is used to get input for both variable & database.
Set :- get input only for variable.

19) Types of Procedures
1) Internal Procedures
2) External Procedures
3) Persistent Procedures
4) Super Procedures

20) Persistent Procedures :- It keeps the value of external procedure using handle variable.

21) How to run external procedures persistently :- Run <external procedure name> persistent set <handle variable name>.

22) Use of parameter :-
The parameter is used to passing the argument to function and Procedure.

23) Types of Parameters :-
1) Input 2) Output 3) Input-output

24) How to Define Parameter :- Define input parameter parameter As Integer no-undo.
Define <parameter type> Parameter <Parameter name> as <datatype> no-undo.

25) Types of programming models :-
1) CUI 2) GUI

Project

IT Request :-
My project name is it request. In my project we are having different fields.

Progress Open Edge

1) Progress 4gl :— openedge advanced business language is a business application development language. It supports multiple programming language & It is used for both front-end and back-end. It is a fourth-generation programming language.

2) Versions :— latest version — 10.1 open Edge.
currently working versions is progress 9.1 D → procedure oriented

3) What are the two ways of programming in progress?
CUI (procedure Editor) → character user interface.
GUI (ADM-2) → Graphical user interface.

4) Syntax of message stmt?
Message " "
View-As Alert-box.

5) Syntax of Run stmt?
Run <internal/external> procedure name.
Eg:- Run calculator.p

how to connect database
by using Schema holder.
Pro env or db.

6) Data types in progress?
1) character 2) integer 3) Decimal 4) logical 5) Date 6) Handle
7) widget-handle 8) com-handle 9) Rowid / Recid

7) Syntax of variable declaration?
Syntax :— Define variable <variable-name> as <data-type> no-undo.
Eg:- Define variable cName as character no-undo.

8) Why we are using no-undo?
A) we are using no-undo while declaring the variables.

9) How many data formats available in progress?
A) Two types
1) MDY 2) DMY
month/date/year date/month/year.

10) looping stmts?
1) Repeat 3) For-each
2) Do 4) Do-while.

Locks – 3 types :-
1) no lock – Read the data
2) Share lock – do not specify any thing the default is share lock.
3) Exclusive lock – update the data.

Trigger :- Event of each it will executed on particular code.
1) Assign 2) delete 3) update 4) create

where clause :- It is point out to the table place.

Appserver → It creates a handle.

No-undo :- Whenever we do modification in the progress objects like variable temp-table the previous value will be stored in before image file. To avoid this we are using no-undo. so, this will increase the performance of a program.

Can-find :- It is a function, cannot bring the data record into memory.

Can-do :-

Buffer copy :- A source record to a target record by copying each source field to the target field of the same name.

Buffer compare :- comparison of two records (source and target) by comparing.

~~Not statement~~ Types of blocks
1) DO 2) For 3) Repeat
4) Internal Procedure
5) Trigger Procedure.

Connect statement
Database connection sports 2020.

~~Export statement~~
Window & Dialog Box :- A dialog-box will not allow user to access parent window until dialog box is closed.

Transaction :- It is one iteration of the statement for each, Repeat or procedure block that contains direct updates to the database.

Prodataset :- It is a collection of temp-tables. Ex :- Customer order orderline.
Ex :- 3 tables club together into one dataset.

work table :- defines a work table (a temp-table stored in memory) for use in one or more procedures, or within a single class.

Free form query :- open the new file in the source view & Right-click the query widget in the outline view and select edit freeform query. The edit freeform query dialog box opens.

Ocx :- timer Controllers – set-Timer internet explorer.

Parameter :- passing the arguments to the function and procedure.

Types of procedures :-
1) Internal 2) External
3) Persistent 4) super

Prompt-for
Get input from both variable & database

Set
Get input only from variable.

Eg:-
```
#include <stdio.h>
void main()
{
int n = 5;
int on = 0;
while (n != 0)
{
  if (n & 1)
  {
    on++;
  }
  n = n >> 1;
}
printf ("\n on = %d\n", on);
}
```

o/p :- 0101 = 5

```
output
n=2
0001
0001
```

Set a particular position bit as a 1:-

```
#include <stdio.h>
int main()
{
int n=5;
int pos=3;
n = n | (1<<pos);
}
```

o/p :- n=5, pos=3

1<<3

```
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
```

```
1 0 0 0
0 1 0 1
1 1 0 1
```

Eg:-
```
#include <stdio.h>
int main()
{
int n=5;
pos=2;
n = n & ~(1<<pos);
}
```

o/p :-

```
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
1 1 0 1 1
```

```
1 1 0 1 1
0 0 1 0 1
0 0 0 0 1
```

Taggle of the particular position:-

```
#include <stdio.h>
int main()
{
int n = 10;
int pos = 2;
n = n ^ (1<<pos);
}
```

o/p :-

```
0 0 0 1
0 0 1 0
0 1 0 0
```

```
0 1 0 0
1 0 1 0
1 1 1 0
```

Set a particular position Zero:-

```
#include <stdio.h>
void main()
{
int n=10;
int pos = 2;
if (n & (1<<pos))
{
printf("position on");
}
else
{
printf ("position off");
}
}
```

if (0) = false

o/p :-

```
0 0 0 1
0 0 1 0
0 1 0 0
0 0 0 0
```

n=10

Eg :-
```
#include <stdio.h>
int main()
{
char n=5;
for (i=0; i<8; i++)
{
  if (n & (1<<i))
  {
    printf ("on position = %d", i);
  }
  else
  {
    printf ("OFF position = %d", i);
  }
}
}
```

Eg :-
```
int main()
int n=5;
int pos = 3;
n = n | (1<<pos3);
}
```

Eg:- $a = (12)_{10} \Rightarrow (1100)_2$

$a << 2 \Rightarrow (48)_{10}$, $a << 1 \rightarrow (24)_{10}$ (12×2)

```
2|12
2|6 - 0
2|3 - 0
  1 - 1
1100
```

0 0 0 0 1 1 0 0 → <<1
0 0 1 1 0 0 0 0 → <<2

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$2^5 + 2^4 = 32 + 16$
$= 48$

2×2×2×2×2 = 32
2×2×2×2 = 16
48

$a << 3 \Rightarrow (96)_{10}$ (12×6)
0 0 0 0 1 1 0 0
0 1 1 0 0 0 0 0 → a<<3
$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$2^6 + 2^5 = 64 + 32$
$= 96$

* whenever we do left shift operation the value will be incremented

Eg:- $a = (12)_{10} \rightarrow (1100)_2$
val → $(12)_{10}$
a<<1 → $(24)_{10}$ (12×2) val×2
a<<2 → $(48)_{10}$ (12×4) val×4
a<<3 → $(96)_{10}$ (12×8) val×8
val ×16
val ×32

Right shift operator(>>) :- whenever we do right shift operation the value will be decreased.

$a = (12)_{10} \rightarrow (1100)_2$

a>>1 = $(6)_{10}$ ⇒ (12/2) val/2 ; 12/2 = 6
a>>1: 0 0 0 0 1 1 0 0 → 0 0 0 0 0 1 1 0 ; $2^2 + 2^1 = 4+2 = 6$

a>>2 = $(3)_{10}$ → (12/4) val/4
a>>2: 0 0 0 0 1 1 0 0 → 0 0 0 0 0 0 1 1 ; $2^1 + 2^0 = 2+1 = 3$

a>>3 = $(1)_{10}$ → (12/8) val/8
(12/16) val/16
val/32
a>>3: 0 0 0 0 1 1 0 0 → 0 0 0 0 0 0 0 1 ; $2^0 = 1$

Eg:- using bitwise operators

```c
#include <stdio.h>
int main()
{
  int a = 10, b = 12;
  printf("\n The value of a = %d", a);
  printf("\n the value of b = %d", b);
  printf("\n the complement of a = %d", ~a);   --> -11
  printf("\n the complement of b = %d", ~b);   --> -13
  printf("\n the value of bitwise And = %d", a&b);   --> 8
  printf("\n the value of bitwise OR = %d", a|b);   --> 14
  printf("\n the value of XOR = %d", a^b);   --> 6
  printf("\n the left shift 2 of a = %d", a<<2);   --> 40
  printf("\n the right shift 2 of b = %d", b>>2);   --> 12
}
```

$2^4 + 2^3 + 2^2 = 16+8+4 = 28$

$\sim n = -(n+1)$

O/P:- a = 10
b = 12

```
  1 0 1 0
^ 1 1 0 0
---------
  0 1 1 0
```

8+4+2 ; 1 0 0 1 ; 8+1 = 9

a<<2: 1 0 1 0 0 0 ; $= 2^5 + 2^3 = 32 + 8 = 40$

0 0 1 1 0 0 ; $2^3 + 2^2 = 8 + 4 = 12$

a = 10
b = 20

a&b
0 0 0 0 1 0 1 0
0 0 0 1 0 1 0 0
----
0 0 0 0 0

a|b
1 1 1 1 0 ($2^4\ 2^3\ 2^2\ 2^1\ 2^0$)
16+8+2+4 = 30

```
2|10
2|5 - 0
2|2 - 1
  1 - 0
1010
```

```
2|20
2|10 - 0
2|5 - 0
2|2 - 1
  1 - 0
10100
```

* when all the components conditions true then only it returns true. otherwise it returns false.

Eg:- a b a&&b
0 0 → 0 1 0 → 0
0 1 → 0 1 1 → 1

Eg:- a=10, b=12 → (1100)$_2$
(1010)$_2$ 1010
(1100)$_2$ a&&b → 1100
1000

5 - 0101
7 - 0111
0101

Truth table

T	T	T
T	F	F
F	T	F
F	F	F

becomes 8 7 6 5 4 3 2 1 0
2 2 2 2 2 2 2 2 2
128 64 32 16 8 4 2 1

2|42
2|6 - 0
2|3 - 0
1 - 1
1000
$2^3 2^2 2^1 2^0$
(8)$_{10}$ decimal value

OR operator (||):- one condition is true then overall condition is true. All conditions are false then only it returns false.

Eg:- n = n||x;
a=10, b=12
a||b = a 1010, b 1100 → (1110)$_2$ → (14)$_{10}$
1010 1100
1110
$2^3 2^2 2^1 2^0$
8+4+2

T	T	T
T	F	T
F	T	T
F	F	F

XOR operator (^):- Both condition came true or false it will be false.
* All operations are '0' or '1' it will be false.

Eg:- a b a^b
1010 1100 (0 1 1 0)$_2$
(a^b) = (6)$_{10}$
1010
1100
0110
$2^3 2^2 2^1 2^0$
4+2 = 6
32 16 8 4 2 1
0 0 0 1 0

T	T	F
T	F	T
F	T	T
F	F	F

T-1
F-0

left shift operator (<<):-
Eg:- 5 0 0 1 0 1 = 5<<2
here 00 it will left shift to '0' only
T-1
F-0
10 0 1 0 1 0
<<1
20 1 0 1 0 0
* we do left shift next result it will be multiplication by 2.

5<<2
8 4 2 1
0 1 0 1
1 0 1 0

Right shift operator (>>):-
Eg:- 5 - 0 0 1 0 1
2 - 0 0 0 1 0
1 - 0 0 0 0 1
5>>2 - Rightshift of 2 means we can do rightshift 2 time.
2|5 (2
2|2 (1
1 (0

one's complement (~):- all zero's are '1', all ones are '0'.
Eg:- ~ 0 0 1 0 1
1 1 0 1 0
a → 0 a → 1
~a → 1 ~a → 0

two's complement:- Awe can take positive number we do 2's complement it will be negative number.

Eg:- char a = 5;
1 byte (8 bits)
128 64 32 16 8 4 2 1
0 0 0 0 0 1 0 1 = 5
1's → 1 1 1 1 1 0 1 0 → 1's complement
0 0 0 0 0 0 0 1 → 2's complement (we add one)
(1) 1 1 1 1 0 1 1 = -5
sign bit

Eg:-
-5 → 1 1 1 1 1 0 1 1
128 64 32 16 8 4 2 1
sign bit (-128)
(-)

-128 + 123 = -5

64 32 16 8 2 1 = 123

~ Eg:- a = (10)$_{10}$
1010
8 4 2 1
~a = (5)$_{10}$ = (1010)$_2$
2|10
2|5 - 0
2|2 - 1
1 - 0
0 1 0 1
$2^3 2^2 2^1 2^0$
$= 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1$
$0 + 4 + 0 + 1 = 5$

Bitwise shift operators:- These are mainly used to move bit values either to left/right positions.

left shift (<<):-
Eg:- a = (12)$_{10}$ → (1100)$_2$
(12×2)
a<<1 = (24)$_{10}$ 0 0 0 1 1 0 0
left shift one position only.
$2^4 2^3 2^2 2^1 2^0$
$2^4 + 2^3 = 16 + 8$ → (24)$_{10}$ 1 1 0 0 — 12

2|12
2|6 - 0
2|3 - 0
1 - 1

Global variable :- The scope of the global variable is the throughout the program.

Eg :-
```
#include <stdio.h>
#include "h.h"
{
extern int a;
printf("%d",a);
}
```
h.h
```
int a=20;
```
by access another file by using extern keyword.

o/p :- 20

* By using extern we can access to multiple files.
* static external & global variable difference is only scope.

Declaration :- In declaration memory will be not allocated for the variable.

Definition :- In definition memory will be allocated for the variable.

Eg :-
```
#include <stdio.h>
void main()
{
int a=10;
int b;
}
```
definition . a=10, b=G.V

Eg :-
```
#include <stdio.h>
void main()
{
int a=10;
static int b;
}
```
definition a=10, b=0

Eg :-
```
#include <stdio.h>
void main()
{
int a=10; — definition
}
extern int b; → declaration.
```

Register :- Scope of register is within the function (or) within block.

* lifetime within the function (or) block.
* Default value is G.v.
* Memory storage is cpu registers.
* There is no space in cpu then it will be stored in stack segment.

Eg :-
```
#include <stdio.h>
void main()
{
register int a;
Printf("%d",&a);
}
```
o/p :- Error.

* There is no address for the cpu registers.

Eg :-
```
void main()
{
int *p;
register int a=10;
p=&a;
}
```
o/p :- Error.

a: 10, 1002

Bitwise Operators :- Binary operators (or), It requires 2 operands * These are mainly used to perform operations on bit levels.

Bitwise & operator (&) :-
Bitwise OR operator (|)
Bitwise XOR operator (^) (or) (Exclusive OR)
leftshift operator (<<)
Rightshift operator (>>)
Once Complement (~) negation (or) bitwise complement

& operator :- Both conditions true then only & operation true.

Eg :-
```
int n=5;
int x=7;
n=n&x;
```
5 → 2-1 → 1-0 → 0-1 : 0101

External static :— The scope of the external static is with in the file.

Eg:
```
#include <stdio.h>
int fun(void);
static int a=10;
void main()
{
 fun();
 printf("%d",a);
}
int fun(void)
{
 int a=30;
 if(1)
{
 printf("%d",a);
}
printf("%d",a);
}
```

o/p :—

file scope
↑
function scope
↑
Block scope

Eg :—
```
#include <stdio.h>
int fun(void);
static int a=10;
void main()
{
 fun();
 printf("%d",a);
}
int fun(void)
{
 if(1)
 {
  printf("%d",a);
 }
}printf("%d",a);
```

Block →
fun
file.

* scope is difference in between static → intern & static external, remaining are same.

Eg :—
```
#include <stdio.h>
int fun(void);
static int a=10;
void main()
{
 fun();
 a=100;
 printf("%d",a);
}
int fun(void)
{
 int a=90;
 if(1)
 {
  a=40;
  printf("%d",a);
 }
printf("%d",a);
}
```

static int a=20;
↓
o/p :—
redefination Error.

o/p :—

life time :- It is throughout the program.

Eg :-
```
#include <stdio.h>
int fun(void);
void main()
{
  fun();
  fun();
  fun();
}
int fun(void);
{
  static int a = 10;
  printf("%d", a);
  a++;
}
```
o/p :- 10
11
12

* Default value of static is '0'.

Eg :-
```
#include <stdio.h>
void main()
{
static int a;
  printf("\n%d", a);
```
o/p :- 0

Eg :- static int a = 10; → initialize → data segment
static int b; → uninitialize.

Eg :-
```
#include <stdio.h>
int *fun(void);
void main()
{
  int *p;
  p = fun();
  printf("%d", *p);
}
static int *fun(void)
{ int a = 10;
  return (&a);
```

Here we will mention static memory is not arised

o/p :- 10

* a is the local variable control go to outside the function the memory will be arised

a	p
10	7000
7000	8000

*p - 10
a - 10
&a - 7000

(1) *: Run time Error.

o/p :- It will be get the segmentation fault problem

Storage classes in C

Storage class	Declaration	Storage	Default Initial value	Scope	life time
1) Auto (A.D) (Automatic) Eg: auto int a=10;	within a function/ block	Memory (RAM)	G.V garbage values	within a function/ within a block	within a function/ block.
2) Extern (external) (Global declaration) Eg:- extern int a;	outside of all functions	RAM	Zero (0)	any file which access that file. (within the file)	Program Runtime
3) Static (Local) (Global)	within the fun/ a block	RAM	Zero	within the fun/ block	within the fun/ block.
4) static (Global)	outside of all functions	RAM	Zero	through out the Program by any function	through Program runtime
5) Register	within a fun/ block	CPU registers	G.V	within the fun/ block	within the fun/ block.

Eg:-
```
#include <stdio.h>
int fun(void);
void main()
{
fun();
}
int fun(void)
{
int a=20;
if(1)
{
printf("\n%d",a);
}
printf("\n%d",a);
}
```
O/P:- 20

life time :- 20

life time of the local variable is with in the function.

Eg:-
```
#include<stdio.h>
int fun(void);
void main()
{
fun();
fun();
fun();
}
int fun(void)
{
int a=10;
printf("%d",a);
a++;
}
```
O/P:- 10 10 10

* Because of the life time of the local variable within the function. whenever control with be go to outside the function. The variable value will be dead. again it will be reinitialize to 10.

* local variable it will be stored in stack segment.
* Default value in local variable is garbage value.

Eg:-
```
#include <stdio.h>
void main()
{
int a;
printf("\n%d",a);
}
```
O/P:- Giv.

Static variable :- Scope of the static is the within the function or within the block.

Eg:-
```
#include <stdio.h>
int fun(void);
void main()
{
fun();
printf("\n%d",a);
}
int fun(void)
{
```
Error

keyword
```
static int a=10;
printf("\n%d",a);
}
```
O/P:- Error

Block Scope :-

Eg:-
```
#include<stdio.h>
int fun(void);
void main()
{
fun();
}
int fun(void)
{
if(1)
{
static int a=1;
printf("\n%d",a);
```
O/P:-

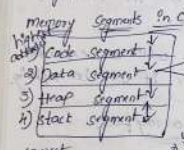
Eg:- #include <stdio.h>
void main()
{
int a=8, b=8, c=10;
c=a==b || ++b==c;
Printf("%d",c);
printf("%d",b);
}
o/p:- 1
8

|| — It checks both conditions.

T
1 ||
one condition is true in || it not checks another condition.

Eg:-

Memory Segments in C:-



lowest address

Storage Classes:- It is mainly used to 1) Variable 2) The location where the variable will be stored 3) Variable Scope 4) Visibility of the variable 5) who can access a variable 6) lifetime

Scope of the local variable:- It is with in the block.

(or) with in the block.

1) Auto (or) local variable
2) Static variable
3) register variable
4) Global (or) Extern variable

Eg:- main()
{ int a;
if(1)
{ int b;
}
}

You can use with in a function only.

You can use with in a block only.

* without the function block we can't use.

use can declare outside the all functions like this throughout the prg

function Scope — It is with in the function

Eg:- #include <stdio.h>
int fun(void);
void main()
{ fun(); → function calling
printf("%d",a);
}
int fun(void) function definition
{ int a=10;
printf("%d",a);
}
o/p:- Error.

Eg:- #include <stdio.h>
int fun(void);
void main()
{
int a=20;
fun();
printf("%d",a);
}
int fun(void)
{ int a=10;
printf("%d",a);
}
o/p:- 10
20

Block Scope:- The scope only with in the block

Eg:- #include <stdio.h>
int fun(void);
void main()
{
fun();
}
int fun(void)
{
if(1)
{
int a=1;
printf("\n%d",a);
}
printf("\n%d",a); → This line error because the scope only with in the block.
o/p:- Error

Eg:- #include <stdio.h>
int fun(void);
void main()
{
fun();
}
int fun(void)
{ int a=20; → with in a fun bly
if(1)
{ int a=1; → with in a block.
printf("\n%d",a);
}
o/p:- 1
20

Eg:- #include <stdio.h>
#include "src.h"
#include "src.h"
void main()
{
printf("\n a=%d", a);
}
o/p:- redefination error.

* we are including more than one time same file it will be redefination error.
* By using header guard we can reduce errors in include two or more files.

Eg:- #ifndef SRC_H
#define SRC_H
int a=20;
#endif

* Condition compiling :- It will be removed in preprocessing time only. It not came to compiling.

Eg:- #include <stdio.h>
void main()
{
printf("welcome to bangalore\n");
printf("welcome to chennai\n");
printf("welcome to pune\n");
#if(0)
printf("welcome to delhi\n");
printf("welcome to hyderabad\n");
#endif
printf("welcome to kadapa\n");
printf("welcome to kerala\n");
}
o/p:- if(0) not come to compiling.
bangalore

Eg:- comments :-

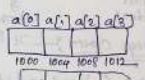
Eg:- #include <stdio.h>
void main()
{
/*printf("welcome to hyd\n");
pf(" " "kerala\n");
pf(" " "bangalore\n");*/
pf(" " "kadapa\n");
// pf(" " "pdt\n");
// pf(" " "del\n");
}
-> after preprocessing
void main()
{
printf(" " "kadapa\n");
}

// - Single line comments.
/* ... */ - multiline comments
/* ... - invalid comments
... */

* we have any comments will be there the comments will be remove at preprocessing time.
* we have any macro is there macro will be expanded.
* we have any file including is there it will be replacing.

without size we can find out size of array :-

Eg:- #include <stdio.h>
void main()
{
int a[];
printf("\n size=%d", (char*)(&a+1) - (char*)(&a));
}
o/p:- 16



1016
1000
0016

Eg:- #include <stdio.h>
void main()
{
int a=2, b=1, c=10;
c=a==b++ || b==c;
printf("%d", c);
printf("%d", b);
}
o/p:- 0
1

False - 0
True - 1
2==1
=0
|| - one condition false total false.

Eg:- #include <stdio.h>
#define SQR(x) (x)*(x) → These are function like macro
void main()
{
int a=3;
printf("%d\n", SQR(a+2)); → (a+2)*(a+2) = (3+2)*(3+2) = 5*5 = 25
}
o/p :- 25

Eg:- Swap two variable by using macro.
#include <stdio.h>
#define swap(a,b) int c; c=a; a=b; b=c;
void main()
{
int x=10;
int y=20;
swap(x,y); → after preprocessing → int c; c=a; a=b; b=c;
printf("%d%d", x, y);
}
o/p :- x=20
y=10

Eg:- Sum of the numbers by using macro.
#include <stdio.h>
#define sum(n) int sum=0; → multiple lines by using macro end of the line we mention.
int r=0;\
while (n!=0)\
{\
r=n%10; → remainder
sum=sum+r;\
n=n/10;\
}\
printf("%d", sum)

void main()
{
int n=123;
sum(n);
}
o/p :- 6

chain macro
Eg:- #define constant PI
#define PI 3.14
constant
PI
3.14

what is the difference between macro & function.

macro
* If we are using the macro the program size will be increased.
* If we are using macro 10 times in our program. The same code will be replacing 10 times. So that the program code will be increasing. But in function if I calling 10 times every time it will go to function defination and resolve it. So program size not increase.

function
* If we are using the functions the program execution speed less compare to macro.
* when ever calling the function the control will go to function defination and again back to calling function. So that it will take more time. So the execution speed is less compare to macro.

File including :- one file will be include to another file is called a file including.
SRC.H :- we want to reuse the functions/ macros in several programs.

Two types of c files:-
.H — header files
.C — source files.

Eg:- ##
int a=20;
SRC.C :- Program written by the programmer must saved with ".c" extension.
Eg:- #include <stdio.h>
#include "src.h" → Src.h → we are include one file to another file.
void main()
{
printf("\n A=%d", a);
}
o/p :- 20

Eg:- #include "src.h"
#include <src.h>

Angular bracket

String
* The compiler searching for the in current directory. If it's not there it will go to standard library.

* A function can call any number of types in a program it is called function calling.

Eg :- funname (args); } function calling.
var = funname (args); }

Eg : sum(x,y); // function calling
x,y → Actual para/ Actual args

Eg :- sum(x,y);
result = sum(x,y); → By using this variable we can call function.

* A function which may not return any value should be declare with the return type void.

Eg :-

* Function parameters :- They are two types.
1) Actual parameters → This parameters which are specified during function calling this parameters are called Actual.
2) Formal parameters

* The parameters which are defined in function declaration (or) function defination are called as formal parameters (or) formal arguments.

Eg :- Sum of two numbers using userdefined function in C.

```
#include<stdio.h>
int sum(int,int);        → function declaration
int main()
{
  int x,y,result;
  printf("\n enter x,y values:");
  scanf("%d%d", &x,&y);
  result = sum(x,y);     → function calling   x,y are called actual arguments.
  printf("\n the addition=%d", result);
}
int sum(int a, int b)    → fun definition  (x=10 y=20)
                           formal arguments
{
  int c; // local variables
  c = a+b;     c = 10+20
  return c;    c = 30
}
```

o/p :- enter x,y values:
10 20
30

Compiling Steps in C :-

```
Source code                  Hi.c
   ↓
Preprocessing
   ↓ ───────────────→ Hi.I  → Intermediate file
compiling
   ↓ ───────────────→ Hi.Asm → Assembly language
assembling
   ↓ ───────────────→ Hi.obj → object file → It is created whenever compile C Program is
linking                                       Successfully Completed then the compiler
(It is a Program)                             Create object code for source code.
     └──────────────→ Hi.exe → execute file.
```

Preprocessing :-

macro :- macro is the one of the preprocessing at precompiling time the macro template will be expanded.

Eg :-
```
#include <stdio.h>
#define X 20        → macro / macro template
void main()
{
  int y=X+10;       → after preprocessing  int y=20+10;
  printf("%d\n",y);
}
```
o/p :- 30

cpp filename.c

operator precedence
C Rules
/
*
+
-

Eg :-
```
#include <stdio.h>
#define SQR(x) x*x    → macro
void main()
{
  int a=3;
  printf("\n%d", SQR(a));   → a*a - preprocessing
}
```
o/p :- 9

Eg :-
```
#include <stdio.h>
#define SQR(x) x*x
void main()
{ int a=3; printf("%d\n", SQR(a+2)); }
```
(a+2)*(a+2)

o/p :- 3+2*3+2 → 11
= 3+6+2
= ~~5*5~~
= 11

Eg:- #include <stdio.h>
int fun(char*, int)
void main()
{
char a[] = {'a','b','c','d'};
int n = sizeof a / sizeof a[0];
fun(a,n);
}
int fun(char *p, int n)
{
int i;
for(i=0; i<n; i++)
{
printf("\n%d", (p+i));
printf("\n%d", *(p+i));
}
}

p[0]	p[1]	p[2]	p[3]
a	b	c	d
1000	1001	1002	1003

o/p :-
1000
a
1001
b
1002
c
1003
d

Eg:- #include <stdio.h>
void main()
{
int a[5] = {10,20,30,40,50};
int i;
for(i=0; i<5; i++)
{
printf("\n i = %d \n", a[i]);
}
}

a[0]	a[1]	a[2]	a[3]	a[4]
1000	1004	1008	1012	1016

o/p :-
10
20
30
garbage values

a[0] = {10,20,30,40,50};
o/p :- 10
garbage
a[] = {};
→ Error

Eg:- #include <stdio.h>
void main()
{
int a[3] = {10,20,30};
printf("\n address = %d", a);
printf("\n address = %d", &a);
printf("\n address = %d", a+1);
printf("\n address = %d", a+1);
printf("\n address = %d", &a+2);
printf("\n address = %d", a+8);
printf("\n address = %d", *a);
printf("\n address = %d", *(a+1));
printf("\n address = %d", (*(a+1))-1);
printf("\n address = %d", *(a+1)-1);

a[0]	a[1]	a[2]
10	20	30
1000	1004	1008

1012	1016	1020

1024	1028	1032

garbage value
1004 - 1001 = 30
1008 - 1004 = 20 - 1 = 19

o/p :-
1000
1000
1012
1004
1024
1008
10
garbage value
30
9

User-defined functions in C :-
* Generally an user-defined fun in C has the following parts
1) function declaration/function prototype
2) fun definition
3) fun calling
4) fun parameters
5) Return statement

returntype functionname (type1 para1, type2 para2, ...);
(or)
returntype function name-identifier (parameter list);
Eg:- int sum(int, int);
(or)
int sum(int a, int b);
semicolon;

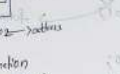
returntype functionname (type1 para1, type2 para2)
{
// local var dec
// instrs to perform task
// return state
}
* fun definition we don't use semicolon.
statements // function body

Eg:- int sum(int a, int b)
{ int c;
c = a+b; return c;
}
formal params
formal args

Call by Reference :- (or) pass by reference
→ means address

```
#include <stdio.h>
int Swap (int *, int *);
void main()
{
  int a=20;
  int b=10;
  Swap (&a, &b);   → I am calling this function through address of variable.
  Printf ("\n a=%d b=%d", a, b);
}
int Swap (int *x, int *y)
{
  int z=*x;
  *x=*y;
  *y=z;
  Printf ("\n x=%d y=%d\n", *x, *y);
}
```

o/p :- x=10 y=20
a=10 b=20



* only pointer variable can store the address of variable.
* * Symbol is used pointer variable.

Pointer :- It is nothing but a variable it will be store the address of another value. variable store value. Pointer store address.

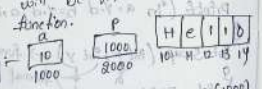
Call by value

* In call by value we are passing the variable values as a argument.
* If as changing variable values in called function the changes will not be reflected in calling function.

Ex :-
```
#include <stdio.h>
void main()
{
  int a=10;
  int *p;
```

Call by Reference

* In call by Reference we are passing address of the variable as an argument.
* If I changing variables in called function the changes will be reflected in called function.

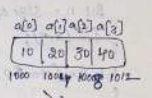


Eg :-
a → 10
&a → 1000
p → 1000
&p → 2000

*p → 10
*(1000) = 10
**p → *(10) = H
**[1000] = *(10) = H

```
p=&a
Printf ("%d %d\n", p, *p);
}
```

Eg :-
```
#include <stdio.h>
void main()
{
  int a[4]={10,20,30,40};
  int i;
  for (i=0; i<4; i++)
  {
  Printf ("\n address=%d\n", *(a+i));
}
```



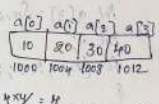
o/p :- address
10
20
30
40

(a+i) with out star
o/p :- 1000
1004
1008
1012

Eg :-

Through function we have to pass the array and print the address :-

Eg :-
```
#include <stdio.h>
void main()
{
  int a[]={10,20,30,40};
  n = size of a / size of a[0];
  fun (a,n);
}
int fun (int *p, int n)
{
  int i;
  for (i=0; i<ten; i++)
  {
  Printf ("%d ", (p+i));
  Printf ("\n %d", *(p+i));
}
```



4*4 = 4

P+a[0]
*(p+a[i])

o/p :- 1000
10
1004
20
1008
30
1012
40

Second smallest element of array :—

```
#include <stdio.h>
void main()
{
  int a[] = {1,10,50,80,3};
  int i=0;
  int small;
  int ssmall;
  if (a[i]<a[i+1])
  {
   small = a[i];
   ssmall = a[i+1];
  }
else
{ small = a[i+1];
  small = a[i];
}
 int n = size of(a)/size of (a[0]);
 for (i=2; i<n; i++)
 {
  if (small >a[i])
  { ssmall = small;
    small = a[i];
  }
  else
  { if (ssmall>a[i])
   {
    ssmall = a[i];
   }
  }
 printf ("\n ssmall = %d", ssmall);
}
```

o/p :— 3

1<1+1
1<2 ✓
small = 1
ssmall = 2

	a[0]	a[1]	a[2]	a[3]	a[4]
int	1	10	50	80	3

1<2 1-3
2-3
2 2<5 1>1

Arrays
- Single dimensional array — 1D array
  - int a [ ]; → 1D
- multi dimensional array. more than one dimension/subscript
  - int a [ ][ ]; 2D Array
  - int a [ ][ ][ ]; 3D
  - int a [ ][ ][ ][ ]; 4D

Functions :— function is the collection (or) block of the statement it will be provided the repeated task.

{ ≡ } → block

```
#include <stdio.h>
int fun(int);
void main()
{
  int a = 10;
  int b = 30;
  fun(a);
  printf ("\nwelcome\n");
  fun(b);
}
int fun(int x)
{
 printf ("\n x=%d\n", x);
}
```

Return type; fun → name of the function; (int) → function arguments; fun(a); → function calling; int fun(int x) → function declaration

o/p :— x=10
welcome
x=30

* The main use of functions is called reusability of code.
* same code we can use number of places.
* large program we can divided into small modules.

Two types of functions :—
1) predefined functions (built in functions)
2) userdefined functions

Eg :— 1) #include<stdio.h> → printf() scanf()
* This headerfiles we use this functions
2) #include <conio.h> → clrscr()
3) #include <math.h> → pow() , sqrt() etc ;
4) #include <string.h> → strrev(), strcat(), strlen()

* we are passing arguments in two ways
1) call by value
2) call by reference

Eg :—
```
#include <stdio.h>
#include <math.h>
main()
{
printf ("\n the squareroot = %f", sqrt(5));
printf ("\n the power fun val = %d", pow (5,3));
}
```

Call by value :— (or) pass by value

```
#include <stdio.h>
int swap(int , int);
void main()
{
  int a=20;
  int b = 10;
  swap (a,b);  → fun calling
  printf ("\n a=%d b=%d", a,b);
}
int swap (int x, int y) → function definition
{ int z=x;
  x = y;
  y=z;
  printf ("\n x=%d y=%d\n", x,y);
}
```

* whenever we are calling a function through the call by value method then the modification which are performed on formal parameters will not effect actual parameters

(a, b) → actual parameter — I am calling this function by call by values.
(x, y) → formal parameter

main: a = 20 (1000), b = 10 (1002)
x = 20 (1004), y = 10 (1006)

o/p :— a=20
b=10
x=10
y=20

Big element size of the array :-

```
#include <stdio.h>
void main()
{
  int a[] = {15,10,50,3};
  int i;
  int big = a[0];
  int n = sizeof(a)/sizeof(a[0]);
  for (i=1; i<n; i++)
  {
    if (big < a[i])
    {
      big=a[i];
    }
  printf ("\n big = %d", big);
}
```

O/P :- big=50

Small element of the array :-

```
#include <stdio.h>
void main()
{
  int a[] = {15,10,50,3};
  int i;
  int small = a[0];
  int n = size of (a)/size of (a[0]);
  for (i=1; i<n; i++)
  {
    if (small > a[i])
    {
      small = a[i];
    }
  printf ("\n small = %d", small);
```

O/P :- 3

Accessing array elements :-

local declaration int a[5]; → array declaration.

a[0]	a[1]	a[2]	a[3]	a[4]
0	1	2	3	4

* we can accessing array elements only through the array index.

Pf ("%d", a[0]) → a[0]

Eg :- int a[3] = {10,20,30};

pf ("\n%d", a[0]); → 10

" " ; a[2] → 30

a[0]	a[1]	a[2]
0	1	2
10	20	30

a[0]	a[1]	a[2]	a[3]
15	10	50	3

* By using loops we can access any number of array elements

Eg :- int a[5] = {10,20,30,40,50};

int i; → loop counter

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

{

Pf("\n%d", a[i]); → a[0] = 10, a[1] = 20, a[2] = 30

}

a[0]	a[1]	a[2]	a[3]	a[4]
0	1	2	3	4
10	20	30	40	50

size - 1 = 5 - 1

i<4

O/P: 10 20 30 40 50

How to enter values into array :-

Ex :- int a[5];

int i;

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

{

scanf ("%d", &a[i]);

}

a[0]	a[1]	a[2]	a[3]
15	10	50	3

Sum of the array :-

```
#include <stdio.h>
void main()
{
  int a[] = {15,10,50,3};
  int i;
  int sum = 0;
  int n = sizeof(a)/sizeof(a[0]);
  for (i=0; i<n; i++)
  {
    sum = sum + a[i];
  }
  printf ("\n sum = %d", sum);
}
```

O/P :- sum = 78

a[0]	a[1]	a[2]	a[3]
15	10	50	3

i :- 0<4

0 = 0+15

sum = 15

i<4

sum = 15+10

= 25

2<4

n=4

Second big element of the array :-

```
#include <stdio.h>
void main()
{
  int a[] = {1,10,150,80,70};
  int i;
  int big = a[0];
  int sbig = 0;
  int n = sizeof(a)/sizeof(a[0]);
  for (i=1; i<n; i++)
  {
    if (big < a[i])
    {
      sbig = big;
      big = a[i];
    }
    else
    { if (sbig < a[i])
      { sbig = a[i];
      }
    }
  printf ("\n sbig = %d", sbig);
}
```

O/P :- sbig = 70

big	sbig
1	0
10	1
50	10
80	50
80	70

5<4 → 2<5

n=5 → i<10

10<50

50<80

80<70

50<70

Pre-increment :— first it will be increment then it will be assign

```
#include <stdio.h>
void main()
{
  int i=6;
  int j=3;
  j=++i;
  printf("\n i=%d j=%d", i,j);
}
```

o/p :— 7,7

i = 6, 7   j = 3, 7

j = ++i

i = 7   j = 7

Eg :— datatype variablename [size]
int a[5];
float b[3];
char ch[10];

Post-increment :— first it will be assign then it will be increment.

```
#include <stdio.h>
void main()
{
  int i=5;
  int j=4;
  j=i++;
  printf("\n i=%d j=%d", i,j);
}
```

o/p :— 5,6

i=5 j=4

j = i++

i = 5

j = 6

Eg :—

```
#include <stdio.h>
void main()
{
  int i=5;
  int j=4;
  j=i+1;
  printf("\n i=%d j=%d", i,j);
}
```

o/p :— 5,6

memory allocation — a = 10; [GV 1000]

a = 20, int a;

Global variable declaration

* A variable which has declare out side of all functions is called as Global variables.

Eg :— int a[5]; [0|0|0|0|0] (0 1 2 3 4)

* local variable stores by default garbage value.

* Global variable stores by default '0'.

int a;

Initialization of Arrays :—

local declaration :— A variable which has be declare within a function or with in a block is called local variable. Eg :— { }

1) Eg :— int a[5]; [GV|GV|GV|GV|GV] (0 1 2 3 4)

2) int a[5] = {10,20,30,40,50}; [10|20|30|40|50] (0 1 2 3 4)

3) int a[5] = {10,15,20}; [10|15|20|0|0] (0 1 2 3 4) we don't use assign that time it store GV.

4) int a[5] = { }; [0|0|0|0|0] (0 1 2 3 4)

5) int a[5] = {0}; [0|0|0|0|0] we use assign that time it stores '0'.

6) int a[3] = {1}; [1|0|0]

Arrays :— An array is the collection of the similar data type. (derived data type)

```
#include <stdio.h>
void main()
{
  int a[4];
  int i;
  for(i=0;i<4;i++)
  {
   scanf("%d",&a[i]);
  }
  for(i=0;i<4;i++)
  {
   printf("\n %d\n",a[i]);
  }
}
```

o/p :— 11
12
13
14

int → data type
a → name of the array
[4] → size of the array — it will store 4 data elements. • Array declaration.

index starts with 0, ends with → size − 1 = 4 − 1 = 3

a[0]	a[1]	a[2]	a[3]
11	12	13	14
1000 (4)	1004 (4)	1008 (4)	1012 (4)

→ block of memory allocation; array location; internal pointer / base address

int a; → primitive data type → we can use only one value

* by using array we can use at a time any number of elements. int a[100];

* A pointer variable will stores the address of a variable (by using * symbol). Eg :— int *ptr;

Formula for array size :—

array size = number of elements * size of the data type.

Eg :—

```
#include <stdio.h>
void main()
{
  int a[4] = {5,2};
  int i;
  for(i=0;i<4;i++)
  {
   printf("%d",a[i]);
  }
}
```

o/p :— 5 2 0 0

* Atleast one element will be required in an array. If there is not it will be print the garbage value. otherwise it will be print '0'.

Array index :—

```
#include <stdio.h>
void main()
{ int a[] = {5,10,20,30};
  printf("\n%d", sizeof(a)/sizeof(a[0]));
}
```

o/p :— 16/4 = 4

Return type of the Scanner :— Scanner will be return number of variables it will be read.

```
#include <stdio.h>
void main()
{
int i,a,b;
if (i= scanf ("%d%d%d", &a, &b))
{
printf ("\n i=%d",i);
}
}
```

o/p :— 2

Swapping :—

```
#include<stdio.h>
void main()
{
int a=10;
int b=20;
int c=a;
a=b;
b=c;
printf ("\n a=%d b=%d",a,b);
}
```

a	b	c
10	20	10
20	10	

o/p :— 20, 10

Addition of without temporary variable Swapping :—

```
#include <stdio.h>
void main()
{
int a=20;
int b=40;
a = a+b;
b = a-b;
a = a-b;
printf ("\n a=%d b=%d", a,b);
}
```

o/p :—

a	b	
20	40	a = 20+40 = 60
60	20	b = a-b = 60-40 = 20
40	20	a = a-b = 60-20 = 40

multiplycation :—

```
#include<stdio.h>
void main()
{
int a=20;
int b=30;
a = a*b;
b = a/b;
a = a/b;
printf ("\n a=%d b=%d",a,b);
}
```

a=	b	
20	30	a = a*b = 20×30 = 600
600	20	b = 20/600 = 20
30	20	a = 30

o/p :— 30, 20

Prime number :—

```
#include <stdio.h>
void main()
{
int n,i,flag=0;
printf ("Enter the n value");
scanf ("%d",&n);
for (int i=2; i<n/2 ; i++)
{
if (n%i==0)
{
flag++;
break;
}
}
if (flag==0)
{
printf ("It's prime number");
}
else
{
printf ("It's not a prime number");
}
}
```

o/p :—

Sum of the number :— Sum of the numbers

```
#include <stdio.h>
void main()
{
int n=123;
int sum=0;
int r=0;
while(n!=0)
{
  r=n%10;
  sum=sum+r;
  n=n/10;
}
Printf("\n Sum = %d\n",sum);
}
```

o/p :- 6

Reverse number :—

```
#include<stdio.h>
void main()
{
  int n=123;
  int rev=0;
  int r=0;
  while(n!=0)
{
   r=n%10;
   rev=(rev*10)+r;
   n=n/10;
}
Printf("\n reverse = %d\n", rev);
}
```

o/p :— 321

Factorial of two numbers :—

```
#include <stdio.h>
void main()
{
  int n=4;
  int fact=1;
  while(n!=1)
  {
   fact = fact*n;
   n--;
  }
Printf("%d", fact);
}
```

n	fact
4	1
3	4
2	12
1	24

1*4 = 4

4! = 4*3*2*1 = 24

o/p :— 24

without Semicolon write one C program :—

```
#include <stdio.h>
  void main()
{
 if(Printf("Hello"))
  {
  }
}
```

if condition
if (0) → false 0

o/p :— hello

Return type of the printf :— printf will be return number of bytes of will be printed :

```
#include <stdio.h>
  void main()
{
 int i;
 if (i=printf("hello"))
  {
   printf("\n i = %d", i);
  }
```

o/p :— hello
i = 5

If-else :— It will be executed both true block and false block statements.

```
#include <stdio.h>
int main()
{
    int marks;
    printf("enter the marks\n");
    scanf("%d", &marks);
    if (marks>=35)        → eg :- 40
    {                       20
        printf("PASS\n");
    }
    else
    {
        printf("FAIL\n");
    }
} o/p :-
```

exit loop - Do-while
Entry loops :- for, while

while loop :— It is used to executed a group of stmts repeatedly until some condition is satisfied.

1) initialization
2) condition
3) body of stmts
4) incre/decre

```
#include <stdio.h>
void main()
{
    int i=0;
keyword ← while (i<5)
    {
        printf("\n i=%d", i);
        i++;  → even numbers
    }
} o/p :- i=0
         i=1
         i=2
         i=3
         i=4
```

* Condition is false it will exit the loop.

i=2, i+2 → odd ↓ i=1, i+2

Eg :-

```
#include <stdio.h>
void main()
{
    int i=5;
    while (i>0)        4>0
    {
        printf("\n i=%d", i);
        i--;
    }    o/p :- 5
}               4
                3
                2
                1
```

For loop :— It is executed repeatedly until condition for satisfied. Condition is false it will exit the loop.

Eg :-

```
#include <stdio.h>
void main()
{
    int i;
keyword ← for (i=5; i>0; i--)   initialization / assignment, condition, increment (or) decrement
    {
        printf("\n i=%d", i);
    }
} o/p :- 5
         4
         3
         2
         1
```

sum = sum + i;
printf("\n sum=%d", sum);
sum program

avg = sum/n;

Do-while Eg :-

```
#include <stdio.h>
void main()
{
    int i;
    for (i=1; i<6; i++)
    {
        printf("\n i=%d", i);
    }
} o/p :- 1
         2
         3
         4
         5
```

Nested for loop :— specifying a for loop within another for loop.

```
#include <stdio.h>
void main()
{
    int i, j;
outer for loop → for (i=0; i<3; i++)
    {
inner for loop → for (j=0; j<3; j++)
        {
            printf("\n i=%d j=%d\n", i, j);
        }
    }   false
} o/p :-
```

i	j
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2

Eg :-

```
#include <stdio.h>
void main()
{
    int i, j;
    for (i=0; i<3; i++)
    {
        for (j=0; j<4; j++)
        {
            printf("\n i=%d j=%d", i, j);
        }
    }
}
```

o/p :-

i	j
0	0
0	1
0	2
0	3
1	0
1	1
1	2
1	3
2	0
2	1
2	2
2	3

Break :—

* It is used to stop the execution process.

* we have to use break statement in any loop along with 'if' construct.
* without if we can't use.

```
#include <stdio.h>
void main()
{
    int i;
    for (i=0; i<10; i++)
    {
        if (i==3)      True 3==3 → stop the execution
        {
            break;  → keyword
        }
        printf("i=%d", i);
    }
}  o/p :- 0
          1
          2
```

Continue :—

* It is used to continue the execution process.
* Condition is false that only not printed after it will continue.

```
#include <stdio.h>
void main()
{
    int i;
    for (i=0; i<10; i++)
    {
        if (i==3)      True 3==3 → this one only not printed
        {
            continue; → keyword
        }
        printf("\n i=%d\n", i);
    }
}
```

o/p :- 0 1 2 4 5 6 7 8 9 (3 is not printed)

C-Programming

Data types :—

- Primary
  - int (4 bytes)
  - char (1 byte)
  - float (4 bytes)
  - Double (8 bytes)
  - short int (2 bytes)
  - unsigned int (4 bytes)
  - unsigned char (1 byte)
- Secondary
  - array
  - strings
  - struct
  - union
  - Enum

in terminal

* Attention Esc+E

Eg :

```
#include <stdio.h>          → Header files  → standard input output libraries
int main()                  → main function
{
  int a;                    → variable declaration   (Data type, variable)
  printf ("enter the a value");   → Print the values
  scanf ("%d",&a);          → Read the printf
  printf ("%d", a);
}
```

* If we have to open the terminal we have to click, Ctrl + Alt + T
* To create the directories the command is mkdir file name.
* To open the directory the command is cd file name.
* To write the program vi file name.
* To write the program we have to click insert 'i', to save the program click Esc. without saving q!. to come outside of the program colon(:wq)
* To compile the program : gcc filename.c
* To run the program ./a.out. → (anyname what we save)
* LS means list of files.

Data types :— The size of the variable should be 31 characters. The variable has capital letters, small letters, character, special characters, numbers etc. But the first should be character.

Function :— They are two functions.

1) user defined functions → Reusability & time saving (user requirement, we can write same code in another time).
2) Predefined functions → Built in functions (already defined) (or) libraries.

* user defined functions run time it should be allocated the memory.
* Predefined function in compile time it should be allocated the memory.
  - printf(), scanf() } stdio.h
  - sqrt(), pow(), sin(), log() } math.h
  - strcat(), strlen(), strrev() } string.h

Eg :—

```
#include <stdio.h>
void main(),
{
  int a = 30;
      a = 50;
  printf ("%d",a);
}
O/P :— 50
```

[30] [50] → we take 2nd memory allocation

```
2) #include <stdio.h>
void main()
{ int a = 30;
      a = 50;
  printf ("%d", a);
      a = 10;
  printf ("%d", a);
}  O/P :— 50
          10
```

```
3) #include <stdio.h>
void main()
{
  int a = 70;
  char a = 30;
  printf ("%d" a);
}
  O/P :— Error
```

Because we don't use same variable in different data types. → Redefinition Error. conflicting types of Errors

```
4) #include <stdio.h>
void main()
{ int a = 70;
      a = 50;
  printf ("%d",a);
}
  O/P :— 50
```

```
5) #include <stdio.h>
void main()
{
  int a = 70;
  int a = 50;
  printf ("%d",a);
}
  O/P :— Error.
```

→ Redefination Errors.

If Condition :—

if(0) → false

any thing is true

Syntax :—

```
if (condition)
   ↓ True
{
  stmts;
}
next stmts;
```

Eg :—

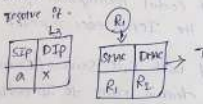
```
#include <stdio.h>
int main()
{
  int marks;
  printf ("enter the marks\n");
  scanf ("%d",&marks);
  if (marks>=35)
  { printf ("PASS\n"); }
```

O/P :— PASS

* It will be executed only True block statement.

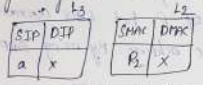
If R_1 which port will be connected that port will be forwarded.

* Once the packet will be reached the 'R_1' it will be remove the 'L_2' header.
* The router will be checking in routing table destination IP address 'x' is connected or not. 'x' is not connected to router 'R_1'.
* Then 'R_1' will be finding the default gateway, default gateway of the 'R_1' is the 'R_2'.
* Now you know the IP address of the 'R_2' but LAN communication should be required the mac address by using the ARP we resolve it.



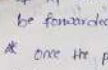
This packet will be created near 'R_1' Router.

* Now the packet will be move from 'R_1' to 'R_2', now 'R_2' will be remove the 'L_2' header.
* Now it will be checking in routing table destination IP address of yes it will be connected.
* Now R_2 know the destination IP address of the x, but LAN communication should be required the mac address.
* By using ARP we can resolve it.



* Now the packet will be move to switch, switch will be checking in MAC table destination mac address of x which port connected to go port number then it will be forward to it will be associated that port will be send.

Eg:- Hub



Problem is broadcast
If traffic C-1 B-1

Switch

Switch is the mac table learns. It means it will learning the mac table

C-3 B-1

C - Collisions
B - Broadcast
because communic

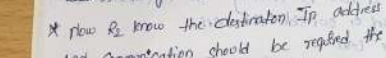
n-1 → C

Collision:- CSMA/CD → Carrier sence multiple Access / Collision detection
CSMA/CA → Collision Avoid.

* Before sending the data it listen the medium. If the medium is idle then only it can send data. If the medium is busy waiting for the random of time again it listen the time.

CSMA/CA:- first I am listen the medium of the medium is idel I am not sending the data waiting for the random of time. and again listen the medium if the medium is idel then only I can send.

Switch:-



12 / 14 / 16 Supporting ports
speed - 10 100 1GB

White Box:- The cards is fixed.
Chassis Box:- easily remove the cards.

* Each port supports 10 to 100 GB.

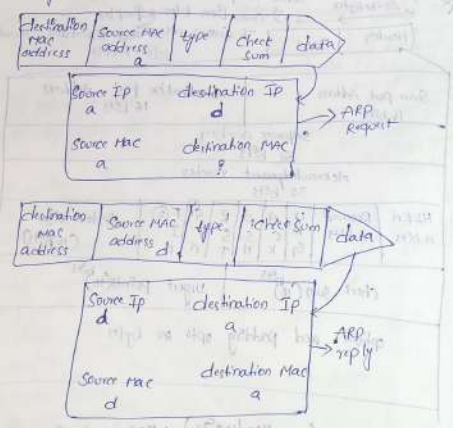
Cisco Switches

Version:- It will be identifying the IP packet which belongs to (Ipv4 or Ipv6)
Total length:- length of the header size + data.
Ipv4 size:- 20 to 40 bytes.
Type of service:- It will provide the quality of service.
Identification:- It will be sequencely identify the IP packets.

1) If I change the value in fragmentation, total length will be change, MF, Fragmentation OFF set will be change.

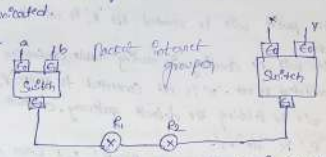
ARP - protocol :— [Address resolution protocol] — By using ARP protocol dynamically mapping from IP address to MAC address.

Eg:— I have the switch then hosts will be connected above. Now I want to communicate from a to d. They are given IP address of d. but LAN communication should be required the mac address. Every hosts has the ARP cache table. The ARP cache table will be stored recently mapping IP address to MAC address. But a is the first communication the entry is not there so that host a will be generated the ARP request. It will be a broadcast message. It will generate ARP request and it will go to every where in LAN. The corresponding destination d reply with the mac address through the ARP reply.



* Once you are receiving the destination mac address now the host a will be updating the ARP cache table then it will be communicated.

Piring :—



* The ping checks whether the network is connected or not
* In ping we use ICMP (Internet control management protocol) protocol. It will be sending the ICMP echo request and expecting the echo reply.
* Here the host will be interact with DNS servers (Domain named system). To resolve the domain name to IP address.
* Here IP address is loop back IP address.

Eg:— 127:0:0:1

* Address will be send with in the system.
* First the host A will be checking the destination address with in the LAN or outside the LAN.
* x is outside the LAN.
* Now the host a will be fetching the default gateway IP address. The default gateway of host a is R_1.
* you know the IP address of R_1 but LAN communication should be required the mac address. By using ARP we can resolve it.
* Now host a will be consider the two headers one is L_2 and one is L_3.

Eg:—

SMAC	DMAC
a	R_1

L_2 header

SIP	DIP
a	x

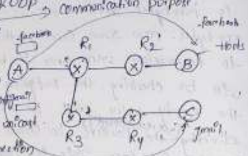
L_3 header

* Now it will be go to the switch. Now switch will be checking in the mac table destination mac address

Transport layer :— It support only unicast (slow process)

Address – port number – 16 bits

* It uses protocols are TCP & UDP → communication purpose
* Address port number size is 16 bits – 2 bytes.



* TCP :– It is a Connection oriented and reliable transport layer protocol. * It supports only unicast.

What is the meaning of connection oriented?

* Before sending the data first we need to establish the connection from client and Server. [TCP is a circuit switching]

How to establish the connection?

1) By using three way handshaking we can establish the connection.

TCP [Transmission Control Protocol]

Client —Syn segment (packet, synchronization)→ Server

← Syn+Ack segment —

— Ack segment →

Bidirection communication (or) full duplex

* client send syn segment sents to server, server thinking for some body will be comming for the request, server will establish the connection for the client. server is sending the syn+ack segment to client then client will be establish the connection for the server and send to acknowledge segment to server.

3) How to close the connection?

1) By using four way handshaking [it can terminate the connection]

FIN segment send the terminate request.

Client —FIN Segment→ Server

← Ack (Acknowledgement) segment —

← FIN segment —

— Ack segment →

Full close

we can connect both at a time

* In connection time some information sharing. Eg:—

1) MSS [Maximum segment size]

* This amount of data only will needed.

2) window size :— Exchanging window size

3) sequence number. It also exchanging.

Eg:— Client —FIN Segment→ Server

← Ack —

half close

TCP will provide the flowcontrol :— Before you are sending the data waiting for the Acknowledgement of the previous data. If we are receiving the Acknowledgement with in the timer explore we can send the next data. If the timer will be expair to Acknowledgement will be not receive we can resend the same data.) (or) poor machanism

Sliding window protocol :— (more)

* Client will be sending the window size of the segment at a time waiting for the acknowledgement of the close segment.

UDP :– It is a connection less and unreliable transport layer protocol. there is no acknowledgment transport.

* It supports both unicast and broadcast.

(fast process) [user datagram protocol] [UDP is a packet switching]

what is protocol?

1) connection b/w 2 peoples.

It follow rules and maintain connection b/w peoples

20-60 bytes

Header	Data

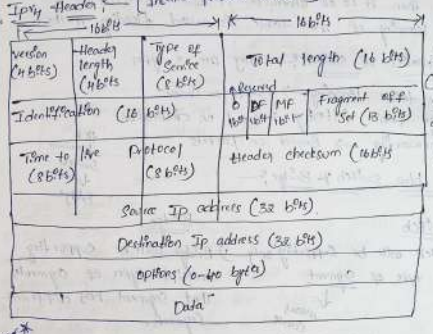
Source port Address 16 bits					Destination Port Address 16 bits			
Sequence number 32 bits								
Acknowledgement number 32 bits								
HLEN 4 bits	Reserved 16 bits	URG	ACK	PSH	RST	SYN	FIN	window size (16 bits)
Check sum (16 bits)					Urgent points (16 bits)			
options. and padding upto 40 bytes								

MSS = MTU – (IPv4 Header size) + TCP header size

= 1500 – (20 + 20) = 1460 — 1460 + 20 + 20 = 1500

* In wan communication we have protocols have IPv4 (or) IPv6.

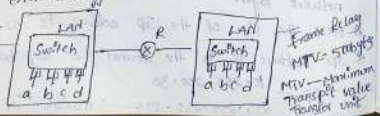
IPv4 Header :— [Internet protocol version 4]



Time to live :— (TTL)

* It will avoid the loops in network layer
* By transfering the data from R_1 to R_2 than R_2 to R_3 then R_3 to R_1 the looping will takes place.
* By using TTL we can avoid the looping. If we give TTL value it will be decrement the value. whenever the packet will go the router.
* when the router will be receive the packet if the TTL value is '1' now the router set the TTL value as '0' and send the timeout error message to the sender.

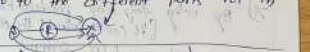
Fragmentation :—



* Let us example I have the two LAN's interconnected through the router one is ethernet LAN the MTU of ethernet is 1500 bytes another one is frame relay LAN the MTU is 500 bytes. Now I am sending the data from ethernet LAN to frame relay LAN. once it will be go to the router it will be checking the outgoing interface empty [routing table] value. If the empty value will be less [compare with] the router it will be dividing the packets in small pieces and send the outgoing interface. This is called as a fragmentation.
* The sequence number is same for all the fragment packets after fragmentation. How to find out the 1st sequence fragmentation based on the fragmentation off set we can find out the sequence.
* If the empty value will be greater the router it will be directly send to outgoing interface.

a) How to know that this is the final fragmentation packet?
a) whenever the MF (more fragmentation) getting '0' then it is a final fragmentation.

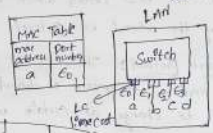
* A packet can be fragmented at several times eg: 100, 100, 100, 100, 100, 100
* Defragmentation can takes place only in destination side. Because destination has different ports. whenever the packet fragment it will be go to the different ports not in same port.

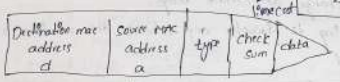


LAN L1	WAN L3	L4
* Data link layer * Here sending data	* Here sending packets * network layer	* Transport layer
* L2 communication MAC address	* Ip address	* port number address 16 bits = 2 bytes
* Mac address format hexadecimal	* Ip address format loop back	* Tcp / UDP Protocols
* Mac size (48 bits - 6 bytes)	* Ip address size (32 bits = 4 bytes)	
* Ethernet protocol, VLAN	* Internet Protocol V4 [IPV4] * IPv6, OSPF	

Switch will be learning the mac table by ——— (or) flooding method.

* Switch doesn't understand IP addresses they only works based on mac address.





Steps to Read ———

* Switch accepts both destination and source address to read.
* First it will be checking the source mac address
* Now entry is not there so it add the entry.
* Next it will be checking the destination mac address.
* Destination 'd' is not there so that it will be forwarded.
* Now sending the data from b to a.
* Switch will be check the source address.
* Entry is not there so it add the entry and forwarding to the table.

d	s
d	a
a	b

Mac-Table

a	E_0
b	E_1

* Now the data will be send from a to d.
* Now switch will be check the source address and destination address. Now entry is not there it add the entry.
* Destination d is not there so that it will be forwarded. Now switch the data from b to a. Mac-table
* If source is already there it will check the port number. If the port number is also same it will be refreshing the timer.

d	s
d	c

a E_0
b E_1
c E_2

* Here the timer is "ageo timer".
* whenever we add the entry the timer will be on.
* It will increase upto the time limit then it will be refreshing the stack.
* If the time will be '0' then it will be remove from the mac-table.

* after removing the switch will be check if the transfer the data from b to a.
* Then a is there and it will check the port number. If it is same then it is ok otherwise it will remove(or) we will rename.
* The learning of mac table is called flooding method.

Functionalities of Switch ——— They are 3 types.

1) Mac table learning → Based on source
2) Mac table filtering → Based on destination
3) Super Forwarding → Based on packets

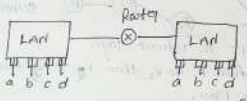
Hub → source
Switch, Router
Bridge

Diff b/w Switch & Bridge ———

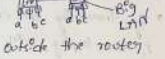
Switch	Bridge
1) Switch will be supporting only one type of Segment (means cable)	1) Bridge will be supporting multiple types of segments. That, Segment has different Segments.

WAN-communications ——— (wide area network)

* More than one LAN interconnected through Router is called wan communication.



Switch – 48bits 6bytes
WAN – 32bits 4bytes



* Any device which will be communicate outside the router than it is called wan communication.
* If the device is not there than it is called WAN.
* Here we have Ip address in WAN communication.
* WAN communication comes under the 3rd layer means network layer.
* Here the size of the Ip address is 32-bits – 4bytes.
* Ip address is in the format of loop back. $2^8 = 256$.

Eg :- 10.255.200.30
255.255.255.255 → max/min Ip address 0-255

Jai Sriram.
5/09-2023

Networking

5 layers — based on TCP/IP stack.

TCP/IP stack :— (Transmission Control Protocol / Internet Protocol).

* Stack will be divided by different layers.

Application layer — L5
Transport layer — L4
Network layer — L3 — WAN
data link layer — L2 — LAN
physical layer — L1

(C++, python — C)

MAC address is fixed — But IP address is not fixed

Network technologies are 3 types :—
1) LAN (local Area network) — (WAN) Can will be comming into data link layer. (small ***)
2) WAN (wide area network) — [To connect more than devices by using internet].
3) MAN (Metropolitan area network) → To connect more than 10 LAN is called man.

LAN :— More than one node (host) (Laptop) interconnected in small area by using L2 devices. These are 3 types. [Connect device by using local area]

1) Switch 2) hub 3) Bridge

* Generally networking has 3 types of address :—
1) unicast 2) broadcast 3) multicast

- unicast → One to one node communication
- broadcast → one node to all
- multicast → one node to many [More than one not all]

* In L2 we need to communicate the another device by using address of destination device.

size → 6 bytes

* In L2 communication MAC (Medium Access Control (48 bits)) should be the connection.
S → D (MAC Address)

Hexadecimal format 0 1 2 3 4 5 6 7 8 9 A B C D E F

0 – 9
A – 10
B – 11
C – 12
D – 13
E – 14
F – 15

* MAC address is in the format of hexadecimal.

Eg :— 02:4b:3c:1a:5d:ae → related to broadcast

* MAC address is fixed for every system.
Eg :— 03:5C:4F:0B:2A

* To change the mac address the netchip has to remove and add the new netchip which is called net card.

* Along the network use protocol. [VLAN]

L2 Protocol
Ethernet Protocol :— The way of transferring the data which is medium to access or transfer the data. The medium will be change at certain ways.

* In networking to transfer any network the medium will be required.
S (address) → D address

* Protocol itself divided into two parts Header → Data

destination (6 bytes)	Source (6 bytes)	type (2 bytes)	Check Sum (4 bytes)	Data
MAC address d	MAC address a			

← Header → ← Data →

6 6 2 4
| 1 | 2 | 3 | 4 | data → The data will be required many.

Header

* Size of Ethernet Protocol is 18 bytes.

* I want to communicate a to d, i need to header size is 18 fill the ethernet header destination is d, source is a after that can attaching the data (source, destination) send to switch.

Eg :— d → my address
Switch → port box
d → friend address

MAC - Table

Mac address	port number
a	E_0
b	E_1
c	E_2
d	E_3

destination device

* Now switch has mac table. In mac table we have mac address & port number.

* Once we packed the net into switch it will check whether the mac address is which port will be connected.

Eg :— [node chain] End device is called node

Difference b/w Switch & hub :—

Switch
* Switch will be checking the in mac table, destination d mac address
1) Switch will be supporting both unicast & broadcast

Hub
each port will be connected now send port forwarding the data.
1) Hub will be supporting only broadcast because hub don't have the mac table so that it does not support the unicast. problem is broadcasting & traffic

d is not there then it will be broadcasting