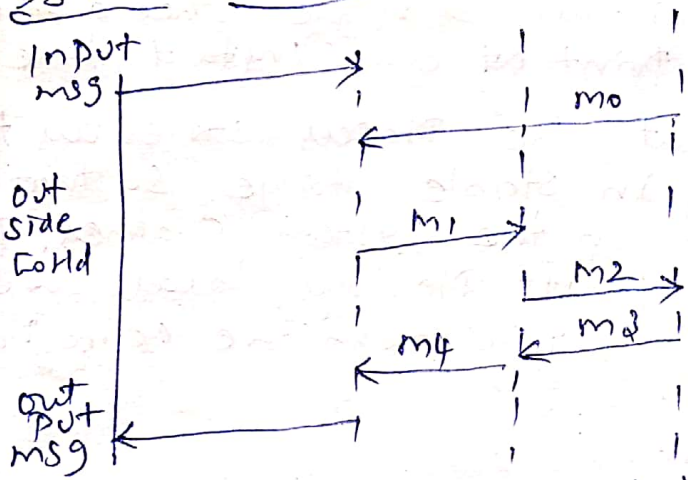
Unit IV

1) Explain about the following (i) System model (ii) A local check point (iii) Consistent system states?

System model:

A set of fixed processes ie $P_1, P_2, \cdots P_n$ that communicate via messages is called a distributed system model. These processes execute a distributed application and communicate with the outside world by sending input messages and receiving output messages.

DS with 3 processes:



In distributed system models, the assumptions like. reliablity of inter processes communications are made by roll back recovery Protocols.

Few protocols assume that the communication sub system is delivered in first out order where as other protocols assume that the communication subsystem can reorder, loose or duplicate the messages.

If any two assumptions, is selected, then it effects the complexity of failure recovery and checkpointing.

If the system recovers properly with consistent internal state with noticeable system behaviour before its failure then this condition is said to be genetic correctness condition for roll back recovery. The protocols of this mechanism maintains the information about (i) internals external interactions.

A Local Check Point:

A local check point is saved state in which all the processes in the distributed system save their local states at specific instance time.

It is defined as the process snapshot state.

The scenario of recording the process state is called as local check pointing.

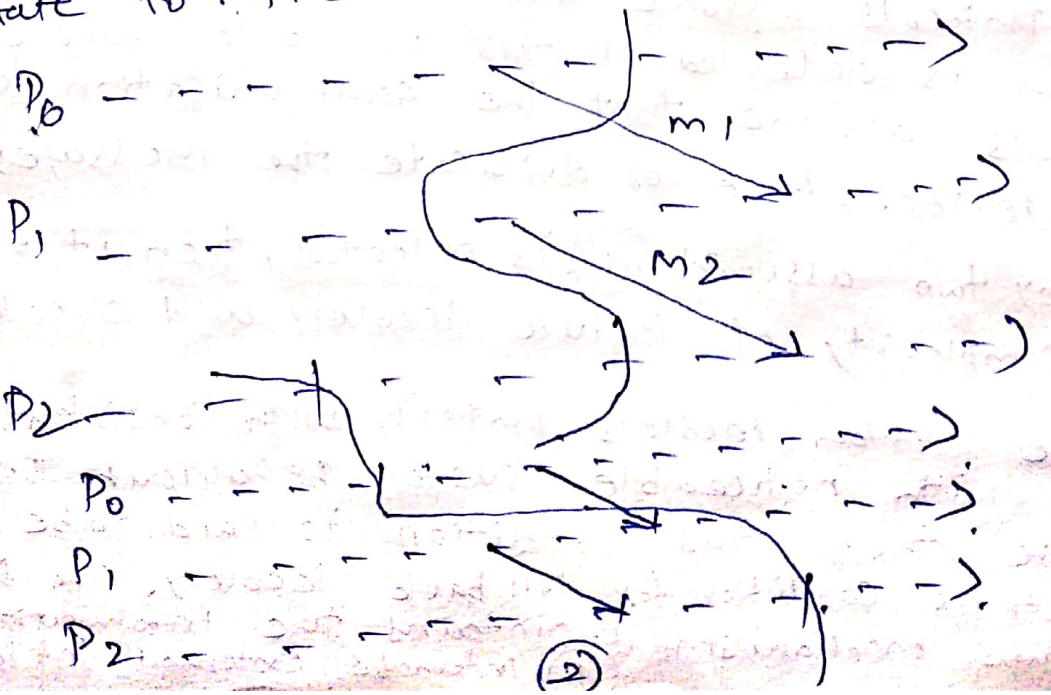
The contents of the check point are based on check pointing mechanism and the application context used.

The process may maintain a single check point or many local check point at any instant time

It is assumed that the process stores all the local check points in stable storage so that they can be recovered when the system crashes. The process may roll back to its previous local check points and hence can restore from the corresponding state.

Consistent State:

A consistent system state is defined as a process state that reflects a message acknowledgement. This effect must cause the corresponding sender state to reflect the message forwarding.



Check Point Based Recovery:

A method which checks the position of each process and communication channel frequently so as to restore them at the time of failure is known as check point based recovery.

The protocols of check point based recovery are less restrictive and easy to implement, because they does not relay on PWD assumptions.

A check point based roll back recovery cannot ensure that after roll back there can be regeneration of the failure execution.

Therefore it is not used for applications that need to interact frequently with outside world.

1. Un Coordinated Check Pointing:

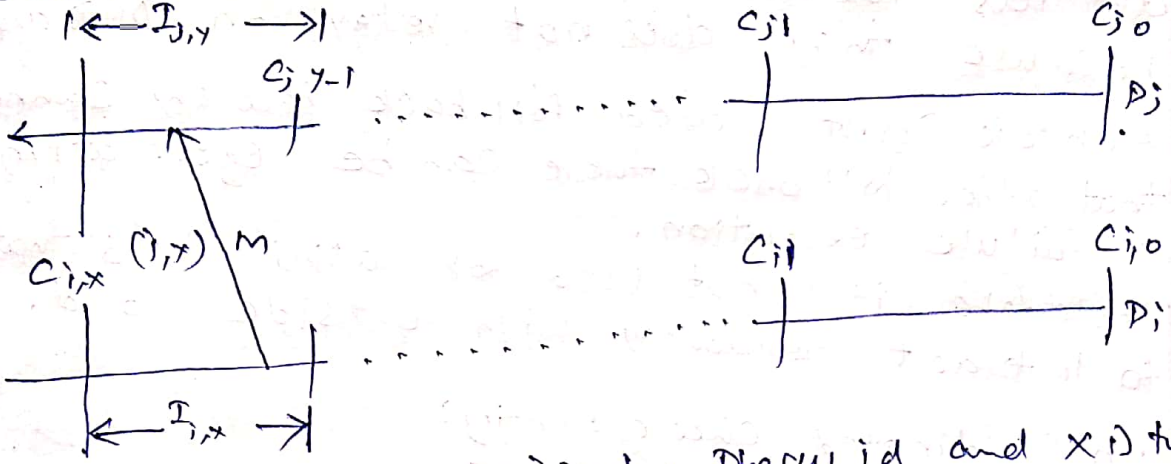
(i) It eliminates synchronization overhead.
(ii) It allows the process to select their check points according to their convenience.
(iii) It provides lower runtime overhead at the time of normal execution.

Uncoordinated check pointing also has the following disadvantages:

(i) There is a possibility of losing huge amount of useful data at the time of data recovery.
(ii) Recovery of data from a failure execution is slow.
(iii) There is no coordination between processes regarding the acceptance of check points. Therefore check points taken by the process can be set as useless check points
(iv) It requires global coordination for computing a recovery line. This coordination has a negative impact over the free selection of the check points.

Every Process at the time of system failure must determine the Global Check Point. While determining the global Check Points all the Processes are required to record the dependencies of their Check points that were taken at the time of failure free operations

An uncoordinated checkpointing make use of direct dependency tracking



P_i & P_j are the process, i is the Process id and x is the checkpoint index, $I_{i,x}$ / $I_{j,y}$ is the checkpoint interval between $C_{i,x}$ and $C_{i,x-1}$ / $C_{i,x}$ and $C_{i,y-1}$

* If a process encounters failures it intiates a rollback by transmitting a dependency request message so that the initiator can easily gather dependency information. maintained by every other process.

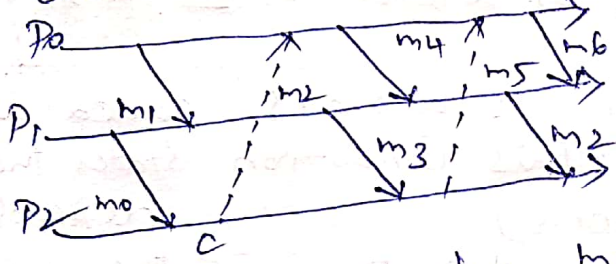
* Any process that receives this request message halts its execution and sends a reply message which contains the dependency information stored on the Stable Storage along with dependency information and again roll back by recovery line request message.

* The process whose current state matches with the request msg resumes the execution and the process whose current state doesnot matches with the request message simply roll back to the previous checkpoint defined by the recovery line.

3) Explain about the deterministic and non deterministic events in the log based rollback recovery.?

* The log based roll back recovery makes excessive use of the fact that a process is executed initially by executing the non deterministic events followed by a sequence of deterministic event time interval.

Here the non deterministic event may be a receipt of message from external (or) internal events of process.



In the above 4 intervals in the execution of a process P_0. The first interval is for creating a process, and other 3 intervals are used to start the receipts of the messages m_0, m_7, m_3.

The process P_0 and the receipt of m_0 uniquely determines the send event of m_2 and hence it can be said that m_2 is a non deterministic event

The simplicity of log based roll back recovery is that, it assumes that the identification of the non deterministic events is easy and their respective determinants can be stored into the stable storage

When a failure occurs, the processes can be recovered using the check points and logged determinants of the non deterministic intervals stored in the stable storage

The No orphan Consistency condition:

Dependency

It defines a set of processes which are influenced by non deterministic event e That is this set includes the processes whose states are depended on event 'e'.

2) Log (e)
It defines a set of processes that logged the determinants of the event 'e' in their volatile storage.

3) Stable e: A process p is said to be orphan when it is not failed by itself.
* A state of process p which is depended on the output of a non deterministic event e and its determinants are not recovered from the volatile memory

$$\forall (e) : \neg \text{stable}(e) \Rightarrow \text{Depend}(e) \subseteq \text{Loge}$$

The above condition is always called as no orphans condition. This condition states that if an ongoing or running process is dependent on non deterministic event e then either e must be logged in the stable storage or the process must have copy of determinant 'e'.
There is an alternative recovery protocol used which guarantees a system which is free from all the orphan processes. This recovery protocol is called as log based roll back recovery protocol.
There are 3 types of log based roll back recovery.
1. pessimistic logging protocol
2. optimistic logging protocol
3. Casual logging protocol.

(4) Explain about Problem definition in Consensus and agreement algorithm?

The fundamental requirement in a distributed system is the agreement between the Processes. This agreement should help in coordinating the Processes to negotiate and exchange the information among them.

Failure model: There are several Processes running in the system, out of which k Processes can be at fault. A faulty Processes is an abnormal Processes that runs in any manner permitted by the failure model. There exists several failure models as send omision and receive omision, fail-stop, Byzantine failures. Each of these models behave in a Different manner.

2. Synchronous / Asynchronous Communication:

If a failure phone process sends a message to some random process say P_1 and fails then it takes a lot of time to figure out the reason of unsuccessful transmission of message in an asynchronous system. While in the synchronous system the failure can be detected easily because the recipient assumes that the expected message might contains some default data and it moves to the next round

3) Network Connectivity: A system supports full logical connectivity. That is each Process in the system can communicate with other Process by directly passing message.

4) Sender Identification:

It should be assumed that the receiver who receives a message can identify the sender. If there are multiple messages expected from a sender in one round then a scheduling algorithm is used to schedule these messages in some default data and it sub rounds.

5) Channel Reliablity:

The channels are always reliable it is the Process that may encounter failure while executing. This is the simplest assumption in the study of agreement algorithms. Even with this simplest assumption there are few agreement Problems that are either solvable or unsolvable.

6) Authenticated Vs non authenticated messages:

In the study of agreement algorithms only unauthenticated messages are dealt. These messages are sent by unreliable sources or faulty Process, which can be tempered or forged. The authenticity of these Process can not be recognized. These messages are also called oral (or) unsigned. messages. To solve the agreement authentication problem the algorithm like digital signature can be used such that the recipient can be recognized any forgery.

7) Agree ment variable: An agreement variable can be a boolean or multi valued variable. It is not necessary for an agreement variable to be an Integer. A boolean variable is used to abstract the values of the algorithms. It has to effect on the other data type results.

(5) Discuss about the Byzantine agreement and other Problems and also Equivalance of Problems and definition?

Byzantine Agreement and other Problems:

This Problem deals with achieving the agreement with other designated Process called Source Process along with its Initial value.

Agreement: This Condition States that Each and every non faulty Processes should agree on the same value.

Validity: This Condition States that if the designated Source Process is non faulty, then the value which is agreed by all the non faulty Processes should be equal to the Initial value of the Source Process.

Termination: This Condition States that every non faulty Process must agree on only one value.

The validity Condition rules is of great Importance. It guarantees that the agreed upon value has a correlation with the source value.

2 other problems in Byzantine agreement:

a) The Interactive Consistency Problem:

This Problem is different from the Byzantine agreement Problem. Here every Process Consists of an Intial value, all the Correct Processes should agree on a group of values such that for each Process has one value.

(i) Agreement: It states that each and every non faulty Process must agree on the same set of values. A [V1, ... Vn].

Validity: It states that of Process say is non faulty with initial value Vi each Process must agree on Vi as the ith element of the array A and if another Process j is found faulty then all the non faulty can agree on any value for array A[j];

3) Termination: It states that all the non faulty Process must consequently decide on the array A.

b) Consensus Problem: This Problem also different from the Byzantine agreement Problem, Here each Process has an initial value and all the correct Processes must agree on the same value.

Agreement It states that each and every Process must agree on the same value.

Validity: It states that if all the Processes consists of same initial value, then the agreed value must be that same value only.

Termination: It states that every non faulty Process must agree on one same value.

Equivalence of the Problems and notations:

The Byzantine agreement Problem, Interactive Consistency Problem and Consensus Problem are equivalent such that the Solution of one Problem can be used as the Solution for the other 2 Problems.

This equivalent Property is used for the reducing one Problem to the other 2 Problems.

The main difference between consensus Problem and agreement problem is that in the Consensus Problem each Process consists of an initial value where as in the agreement Problem only single Process consists of an initial value.

6) Explain about the agreement in synchronous system with failures?

The agreement in synchronous system with failures comes in 2 types.

* Consensus algorithm for crash failure in synchronous system
* Consensus algorithm for Byzantine failures in synchronous system.

① Consensus Algorithm for crash failures in synchronous system:

N processes with upto f (fail stop) Process where $n > f$ for Process P_i $(1 \le i \le n)$

Algorithm:

```
local variable
integer : x ← local value
global Constants:
integer : f;
```

The Process P_i $(1 \le i \le n)$ runs the consensus algorithm upto f crash failures.

```
For round from 1 to f+1
do
if the current value of x is not broad
                              casted.
then
broadcast (x);
y_j ← The value received from the Process.
j in this round;
x ← min ∀j(x, y);
```

Output: x is a consensus value.

Each Process has x_i as an initial value. If f failures need to be tolerated then the algorithm must have f+1 rounds. The Process P_i

forwards the value of its variable x_i to all processes in every round only when the value is not forwarded earlier.

1. Agreement Condition: This condition is satisfied when there exists atleast one round with no process failures in the $f+1$ rounds. In this round, all the processes may not have failed in broadcasting their values and considers only the minimum values broadcast that are received in the round. Hence, the local values at the end are same ie x_i is same for all the non failed processes.

2) Validity Condition: This condition is satisfied when the processes do not forward any fictitious value in the failure model. If the initial value is similar for all i then only value which has been agreed on the agreement condition is forwarded by any of the process.

3. Termination condition: If the two conditions are satisfied then this condition is also satisfied.

Complexity:

The process forwards its value to another process before the failure. In the next rounds the single process having minimum value also manages to broad the value to another process before the occurance of failure.

Lower Bound on the No. of rounds:

A single processes may fail in each round and with $f+1$ rounds, there will be atleast one round with no process failure. which compute the function of the received value to achieve an agreed value.

7) Explain about Phase king algorithm:

Polynomial in synchronous system:

This algorithm operates only in f+1 phases where every phase contains two rounds. It has a special process which plays crucial asymmetrical role as a leader. Hence the algorithm is named as phase king algorithm.

Input variables:

v - boolean
f - Integer
v ← initial value.
f ← maximum number of faulty processes.
f < [n/4];

1. The following f+1 phases are executed by every process. where f < [n/4];
2. For Phase = 1 to f+1
3. do
4. Execute following actions in round 1.
5. Broad cast v to all processes;
6. await value v_j from each process P_j
7. set majority as the value between v_j that occurs > n/2 times.
8. set mult as the total number of times that majority occurs;
9. execute the following actions in round 2.
10. If i = phase
11. then
12. broad cast majority d to all processes;
13. Receive tiebreaks from Pphase.
14. If mult > n/2 tf.
15. then.

12. broadcast majority a to all Processes;
13. Receive the bleaks from P phase.
14. If mult >n/2 f
15. then
16. set V as majority.
17. else.
18. set V as tie breaker.
19. If Phase = f+1.
Do then
21. Display V as output (decision value)

output: The decision value V is the output

Round 1: In each phase of this round every Process delivers the estimation of its consensus to all processes and waits for the values to be broad casted by other Processes. The total number of votes for 0 and 1 are counted at the end of the round. If any of the value is greater than n/2, then its majority variable is set to the consensus value and multi is set to the no. of votes that are received for the majority value.

Round 2: The Phase King for Phase k has the identifier P_k for $k \in \{1 \ldots n\}$. The majority value of Phase king is broad casted is acts as tie breaker for other Processes that have multi values less than n/2 + f. If the Processes receives the breaker from the phase king, it updates the estimation of the decision variable V to the value that is forwarded by the phase king when its multi variable is less than n/2 + f.

UNIT-5.

1. Explain about File.
2. How to read a content from file and display it.
3. How to copy a file from one file to another.
4. Explain Sequential access file.
5. " Random " "
6. C program to ~~store~~ find average of stored number in sequential access file.
7. C program to print names of all files present in the directory.
8. Explain file manipulation function
   a) rename()
   b) freopen()
   c) remove()
   d) tmpfile()
   e) fflush()
9. Explain Command line Arguments?

UNIT-5
File Processing.

Files – Types Of file Processing : Sequential access, Random access : Sequential access file : Egm Pgm – Finding average of no stored in Sequential access file – Random access file – Eg Pgm:- Transaction processing using random access file – Command line arguments.

Files:-

In C programming, file is a place on your physical disk where information is stored.

Why Files are needed ?

*.) When a program is terminated, the entire data is lost.

↳ storing in a file will preserve your data even if the pgm terminates.

*.) If you have to enter a large number of data, it will take a lot of time to enter them all.

*.) ↳ However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.

*.) You can easily move your data from one computer to another without any changes.

Types of Files:-

→ There are two types :

1. Text files
2. Binary files.

(1.) Text Files :-

⇒ Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

⇒ When you open those files, you'll see all the contents within the file as plain text.

↳ you can easily delete or edit the contents.

⇒ They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

(2.) Binary Files

→ Binary files are mostly the .bin files in your computer

→ It store file in the binary form (0's and 1's)

→ They can hold higher amount of data, are not easily readable and provides a better security than text files.

File Operations

In C, you can perform four major operations on the file, either text or binary.

1. Creating a new file
2. Opening an existing file
3. closing a file
4. Reading from and writing information to a file.

Working with Files

↳ When working with files, you can need to declare a pointer of type file.

↳ This declaration is needed for communication between the file and program.

Syntax:-

```
FILE *fptr;
```

Opening a file - for creation and edit.

⇒ Opening a file is performed using the library function in the "stdio.h" header file : fopen().

⇒ Syntax for opening a file in standaud Ilo is :

```
ptr = fopen("fileopen", "mode");
```

For eg:-

```
fopen("E:\\cprog\\newprog.txt", "w");
fopen("E:\\cprog\\oldprog.bin", "rb");
```

⇒ Let's suppose the file newprog.txt doesn't exist in the location E:\\Cprog.

↳ The first function creates a new file named newprog.txt and opens it for writing as per the mode 'W'.

↳ The writing mode allows you to create and edit (overwrite) the contents of the file.

⇒ Now let's suppose the secondary binary file oldprog.bin exists in the location E:\\Cprog.

↳ The second function opens the file for reading in binary mode 'rb'.

↳ you can not write in to the file, reading mode is used only to read the file.

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file.
r	Open for reading	If the file does not exist, fopen() returns NULL
rb	Open for reading in binary mode	If the file does not exist, fopen() returns NULL.
W	open for writing	If the file exist its contents are overwritten. If the file does not exist, it will be created.
Wb	open for writing in binary mode	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file	If the file does not exists, it will be created.

File Mode	Meaning of Mode	During Inexistence of file
ab	open for append in binary mode i.e Data is added to end of file.	If the file does not exists, it will be created.
r+	open for both reading & Writing	If the file does not exists, fopen() returns NULL.
rb+	open for both reading & Writing in binary mode.	If the file does not exists, fopen() return NULL
W+	open for both reading and writing	If the file exists, its content are overwritten. If the file does not exist, it will be created.
Wb+	open for both reading & Writing in binary mode.	If the file exists, its content are over written. If the file does not exist, it will be created.
a+	open for both reading & appending	If the file does not exists, it will be created.
ab+	open for both reading & appending in binary mode	If the file does not exists, it will be created.

CLOSING A FILE

→ The file (both text and binary) should be closed for reading / Writing.

↳ closing a file is performed using library function fclose().

Syntax:-

```
fclose(fptr);
```

//fptr is the file pointer associated with file to be closed.

Sequential access

↳ 1. Reading & Writing to Text File

↳ 2. Reading & Writing to Binary File.

Sequential FILE ACCESS

Reading and Writing to a text file.

For reading and writing to a text file, we use the functions `fprintf()` and `fscanf()`

→ They are just the file version of printf() & scanf().

→ The only difference is that, fprintf() and fscanf() expects a pointer to the structure FILE.

Example 1: Writing to a text file using fprintf()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int num;
  FILE *fptr;
fptr=fopen("C:\\prog.txt",
                "w");
if(fptr==NULL)
  {
    printf("Error!");
    exit(1);
  }
printf("Enter num:");
scanf("%d",&num);
fprintf(fptr,"%d",num);
fclose(fptr);
 return 0;
}
```

⇒ This program takes a number from user and stores in the file prog.txt.

⇒ After you compile and run this program, you can see a text file prog.txt created in C drive of your computer.

↳ When you open the file, you can see the integer you entered.

Example 2

Read from a text file using fscanf()

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int num;
   FILE *fptr;
if ((fptr=fopen("C:\\prog.txt",
                        "r"))==
                    NULL)
   {
    printf("Error! opening file");
    exit(1);
   }
 fscanf(fptr,"%d",&num);
 printf("Value of n=%d",num);
 fclose(fptr);
  return 0;
}
```

⇒ This pgm reads the integer present in the prog.txt file and prints it onto the screen.

⇒ If you successfully created the file from Example 1, running this program will get you the integer you entered.

Other functions like fgetchar(), fputc() etc can be used in similar way.

Reading and Writing to a binary file

Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

↳ To write into a binary file, you need to use the function fwrite().

↳ The function takes four arguments:

1. Address of data to be written in disk
2. Size of data to be written in disk
3. Number of such type of data
4. pointer to the file where you want to write.

Syntax :-

```
fwrite(address_data, size_data,
    numbers_data, ptr_to_file);
```

Example 3
Write to a binary file using fwrite()

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

struct threeNum
{
   int n1, n2, n3;
};
int main()
{
  int n;
  struct threeNum num;
  FILE *fptr;
if((fptr = fopen("C:\\prog.bin",
                 "wb")) == NULL)
  {
    printf("Error in opening");
    exit(1);
  }
  for(n=1; n< 5; ++n)
  {
    num.n1 = n;
    num.n2 = 5*n;
    num.n3 = 5*n+1;
fwrite(&num, sizeof(struct threeNum),
      1, fptr);
  }
  fclose(fptr);
  return 0;
}
```

=> In this pgm, you create a new file prog.bin in the C drive.

-> We declare a structure threeNum with three numbers - n1, n2 and n3, and define it in the main function as num.

-> Now, inside the for loop, we store the value into the file using fwrite().

-> The first parameter takes the address of num

-> The second parameter takes the size of the structure threeNum

-> Since, We're only inserting one instance of num, the third parameter is 1.

-> Last parameter fptr points to the file We're storing the data

Finally, we close the file.

Syntax :-

```
fwrite(address_data, size_data,
    numbers_data, ptr_to_file);
```

Example 3
Write to a binary file using fwrite()

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

struct threeNum
{
   int n1, n2, n3;
};
int main()
{
  int n;
  struct threeNum num;
  FILE *fptr;
if((fptr = fopen("C:\\prog.bin",
                 "wb")) == NULL)
  {
    printf("Error in opening");
     exit(1);
  }
  for(n=1; n< 5; ++n)
  {
     num.n1 = n;
     num.n2 = 5*n;
     num.n3 = 5*n+1;
fwrite(&num, sizeof(struct threeNum),
    }      1, fptr);
  fclose(fptr);
  return 0;
}
```

=> In this pgm, you create a new file prog.bin in the C drive.

-> We declare a structure threeNum with three numbers - n1, n2 and n3, and define it in the main function as num.

-> Now, inside the for loop, we store the value into the file using fwrite().

-> The first parameter takes the address of num

-> The second parameter takes the size of the structure threeNum

-> Since, We're only ~~intera~~ inserting one instance of num, the third parameter is 1.

-> Last parameter *fptr points to the file We're storing the data

Finally, we close the file.

Reading from a binary file.

→ function `fread()` also take 4 arguments similar to `fwrite()` function

Syntax :-

```
fread (address_data, size_data,
       numbers_data, ptr_to_file);
```

Example 4
Read from a binary file using fread()

```
#include<stdio.h>
#include<stdlib.h>
struct threeNum
{
   int n1, n2, n3;
};
int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;
if ((fptr = fopen("C:\\prog.bin",
 {                 "rb")) == NULL)
   Printf("Error in opening file");
   exit(1);
 }
 for(n=1; n<5; ++n)
 {
 fread (&num, sizeof(struct threeNum),
        1, fptr);
 printf("n1: %.d\t
        n2: %.d\t
        n3: %.d\t", num.n1,
          num.n2, num.n3);
  }
  fclose(fptr);
  return 0;
}
```

⇒ In this pgm, you read the same file prog.bin and loop through the records one by one.

→ In simple terms, you read one `threeNum` record of `threeNum` size from the file pointed by `*fptr` into the structure `num`.

→ You'll get the same records you inserted in Example 3.

fgets()

→ The fgets() function stands for file get string.

→ This function is used to get a string from a stream.

Syntax :-

```
char *fgets(char *str,
            int size,
            FILE *stream);
```

→ fgets() function terminates as soon as it encounters either a new line character, EOF or any error.

```
#include <stdio.h>
int main()
{
   FILE *fp;
   char str[80];
   fp=fopen ("ABC.txt","r");
   if(fp==NULL)
   {
     printf("Error");
     exit(1);
   }
   while(fgets(str,80,fp)!=NULL)
   printf("\n%s",str);
   printf("\n File Read. Now
          closing the file");
   fclose(fp);
   return 0;
}
```

o/p :-

```
Abcdee fghi Mno
ffdee
File Read. Now closing the file.
```

fgetc()

=> fgetc() reads a single character from the current position of a file.

Syntax :-

```
int fgetc(FILE *stream);
```

```
#include<stdio.h>
main()
{
   FILE *fp;
   char  str[80];
   int i,ch;
  fp=fopen("prog.c","r");
  if(fp==NULL)
  {
    printf("FILE can't
            open");
    exit(1);
  }
 //Read 79 characters and
//Store them in str

  ch=fgetc(fp);
for(i=0;(i<79)&&(feof(fp)==0);
                          i++)
 {
   str[i]=(char)ch;
   ch=fgetc(stream);
    //reads char by char
 }
str[i]='\0';
 //append the string will
//null characte

printf("\n %s", str);
fclose(fp);
}
```

Reading Data from files

i) fscanf()
2) fgets()
3) fgetc()
4) fread()

Writing Data to Files

1. fprintf()
2. fputs()
3. fputc()
4. fwrite()

fputs()

⇒ The opposite of fgets() is fputs().

→ The fputs() function used to write a single a line to a file.

Syntax :-

```
int fputs(const char *str,
          FILE *stream);
```

```
#include<stdio.h>
main()
{
  FILE *fp;
  Char feedback[100];
fp= fopen("Comments.txt",
                "w");
if(fp==NULL)
{
  printf ("Error");
  exit(1);
 }
printf(" Provide feedback:");
gets(feedback);
 fflush(stdin);
fputs(feedback, fp);
fclose(fp);
}
```

o/p:-

provide feedback:
good.

fputc()

→ opposite of fgetc()
→ used to write a character to a stream.

Syntax :-

```
int fputc(int c, FILE *stream);
```

```
#include<stdio.h>
main()
{
  FILE *fp;
  char feedback[100];
  int i;
fp=fopen("comments.txt",
                "w");
if(fp==NULL)
 {
  printf ("Error");
  exit(1);
 }
```

```
printf("\n Proride feedback:");
gets(feedback);

for(i=0; i< feedback[i]; i++)
fputc(feedback[i], fp);
fclose(fp);
}
```

o/p:-

Proride feedback: Good

Random File Access

Getting data Using fseek()

⇒ If you have many records inside a file and need to access a record at a specific position,

You need to loop through all the records before it to get the record.

⇒ This will waste a lot of memory and operation time.

An easier way to get to the required data can be achieved using fseek().

→ As the name suggests, fseek() seeks the cursor to the given record in a file.

Syntax :- fseek()

```
fseek(FILE *stream,
      long int offset,
      int whence);
```

↳ The first parameter stream is the ptr to the file.

↳ The second parameter is the position of the record to be found,

↳ The third parameter specifies the location where the offset starts.

Different Whence in fseek

Whence	Meaning
~~SEEK~~	
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location of the cursor in the file.

Example 5

fseek()

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum
{
  int n1,n2,n3;
};
int main()
{
  int n;
  struct threeNum num;
  FILE *fptr;
if((fptr= fopen("C:\\prog.bin",
            "rb"))==NULL)
 {
   printf(" Error!opening file");
    exit(1);
 }
//moves the cursor to the end of file
fseek(fptr, -sizeof(struct threeNum),
            SEEK_END);

for(n=1; n<5; ++n)
{
fread(&num,sizeof(struct threeNum),
          1, fptr);
printf("n1:%d\t n2:%d\t
        n3:%d", num.n1,
            num.n2, num.n3);
fseek(fptr, -2*sizeof(struct
          threeNum), SEEK_CUR);
}
fclose(fptr);
return 0;
}
```

=> This program will start reading the records from the file prog.bin in the reverse order (last to first) and prints it.

RANDOM ACCESS TO FILE

→ If we want to access a particular record, C supports these functions for random access file processing.

1. fseek()
2. ftell()
3. rewind()

fseek():

This fn is used for seeking the pointer position in the file at the specified byte.

Syntax:

fseek(file pointer, displacement, pointer position);

where;

file pointer → It is the pointer which points to the file.

displacement → It is +ve (or) -ve. This is the number of bytes which are skipped backward (-ve) or forward (+ve) from the current position.

pointer position → This sets the ptr position in the file

value	pointer position
0	Beginning of file
1	Current position
2	End of file.

Examples :-

1) fseek(p, 10L, 0)

0 means pointer position is on beginning of the file, from this statement ptr position is skipped 10 bytes from the beginning of file

2) fseek(p, 5L, 1)

1 means current position of ptr position. From this statement pointer position is skipped forward from current position.

3) fseek(p, -5L, 1)

From this statement pointer position skipped 5 bytes backward from the current position.

ftell()

This function returns the value of the current pointer position in the file.

The value is count from the beginning of the file.

Syntax:

```
ftell(fptr);
```

where, fptr is a file pointer.

rewind()

This function is used to move the file pointer to the beginning of the given file.

Syntax.

```
rewind(fptr);
```

Pgm

To read last 'n' characters of the file using appropriate function (use fseek() & fgetc()).

```
#include<stdio.h>
void main()
{
  FILE *fp;
  Char ch;
fp=fopen("file1.c", "r");
if(fp==NULL)
  {
   printf("Error);
   exit(1);
   }
else
{
  printf("Enter values");
  scanf("%d,&n);
  fseek(fp,-n,2);
while((ch=fgetc(fp))!=
 {                    EOF)
   printf("%c\t",ch);
  }
}
fclose(fp);
 getch();
}
```

FILE Manipulation Function.

a) rename( )
b) freopen( )
c) remove( )
d) tmpfile( )
e) fflush( )

rename() function in C

a Syntax:-

```
int rename(const char
                *oldname,
           const char *newname);
```

⇒ rename() function is defined in stdio.h header file.

→ It renames a file or directory from oldname to newname.

→ It accepts two parameter oldname and newname which is pointer to constant character.

→ It returns zero if file renamed successfully. Otherwise non-zero integer.

Pgm:- for rename()

```
#include<stdio.h>
int main()
{
 char oldname[100], newname[100];

printf("Enter old file path:");
scanf("%s", oldName);

printf("Enter new file path:");
scanf("%s", newName);

if(rename(oldname, newname)==
                          0)
{
  printf("File renamed
          successfully");
}
else
 {
   printf("Unable to rename");
 }
 return 0;
}
```

freopen()

↳ The freopen() function associates an existing streams with different file.

↳ Syntax :-

```
FILE *freopen(const char
              *fname, const
        char  *mode,
              FILE *stream);
```

=> When freopen() function is called, then it first tries to close a file that may be currently associated with stream.

eg :-

```
#include<stdio.h>
void main()
{
  FILE *fp;
  printf("This text is Nice");
                redirected to stout");
fp=freopen("file.txt","wt",
                    stdout);
printf("This text is Nice
          redirect to file.txt");
fclose(fp);
}
```

remove() in C

remove() deletes the given filename.

Syntax :-

```
int remove(const char
           *filename);
```

Pgm

```
#include<stdio.h>
int main()
{
   int ret;
   FILE *fp;
  char filename[] = "file.txt";
fp=fopen(filename,"w");
fprintf(fp,"%s","This is
          tutorial.com");
fclose(fp);
ret=remove(filename);
if(ret==0)
 {
  printf("file deleted");
 }
else
 {
  printf("Unable to delete");
 }
return(0);
}
```

tmpfile()

→ tmpfile() function is defined in "stdio.h" header file.

→ The created temporary file will automatically be deleted after the termination of pgm

Syntax:

```
FILE *tmpfile(void)
```

```
#include<stdio.h>
int main()
{
  char str[] = "Hello";
  int i=0;

  FILE *tmp = tmpfile();

  if(tmp==NULL)
  {
    puts("unable to Create");
    return 0;
  }

  puts("Temp file created");
  while(str[i] != '\0')
  {
    fputc(str[i], tmp);
    i++;
  }
  rewind(tmp);
  while(!feof(tmp))
    putchar(fgetc(tmp));
}
```

fflush()

→ Its purpose is to clear (or flush) the o/p buffer.

→ used for output stream only.

Syntax:

```
fflush(FILE *ostream);
```

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  char str[20];
  int i;
for(i=0; i<2; i++)
{
  scanf("%s", str);
  printf("%s", str);
  fflush(stdin);
}
return 0;
}
```

Examples Program.

1. Write a C program to read name and marks of n number of students from user and store them in a file.

```
#include<stdio.h>
int main()
{
  char name[50];
  int marks, i, num;
  printf("Enter number
          of students:");
  scanf("%d", &num);
  FILE *fptr;
fptr=(fopen("C:\\student.txt",
          "w"));
 if (fptr==NULL)
 {
   Printf("Error!");
   exit(1);
 }
for(i=0; i<num; ++i)
{
 printf("For students %d \n
       Enter name:", i+1);
 scanf("%s", name);
 printf("Enter marks:");
 scanf("%d", &marks);
fprintf(fptr, "\nName: %s
       \n Marks=%d\n",
       name, marks);
}
fclose(fptr);
return 0;
}
```

2. Write a C program to reads name and marks of n number of students from user and store them in a file. If the file previously exists, add the information of n students.

```
#include<stdio.h>
int main()
{
  char name[50];
  int marks, i, num;
printf("Enter no of students");
scanf("%d", &num);
 FILE *fptr;
fptr=fopen("C:\\student.txt",
          "a"));
```

```
if(fptr == NULL)
{
   printf("Error");
   exit(1);
 }
for(i=0; i<num; ++i)
 {
printf("For student%d\n
        Enter name:",i+1);
 scanf("%s", name);
 printf("Enter marks");
 scanf("%d", &marks);
fprintf(fptr, "\nName:%s
        \nMarks=%d\n",
          name, marks);
 }
 fclose(fptr);
 return 0;
}
```

3. Write a C program to write all the members of an array of structures to a file using fwrite(). Read the array from the file and display on the screen.

```
#include<stdio.h>
struct student
{
  char name[50];
   int height;
};
int main()
{
 struct student stud1[5],
                stud2[5];

 FILE *fptr;
 int i;

fptr = fopen("file.txt","wb");

for(i=0; i<5; ++i)
{
  fflush(stdin);
  printf("Enter name:");
  gets(stud1[i].name);

 printf("Enter height");
 scanf("%d", &stud1[i].height);
}

fwrite(stud1, sizeof(stud1),1,
                fptr);
fclose(fptr);

fptr = fopen("file.txt","rb");
fread = (stud2, sizeof(stud2),1,
                fptr);
for(i=0; i<5; ++i)
 {
  printf("Name:%s\nHeight:%d
          stud2[i].name, stud2[i].
                height);
 }
fclose(fptr);}
```

(4) Write a sentence to a file using fprintf() statement

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 char sentence[1000];
 FILE *fptr;

fptr=fopen("program.txt","w");
if(fptr==NULL)
{
   printf("Error");
   exit(1);
}

printf("Enter a sentence\n");
gets(sentence);

fprintf(fptr,"%s",sentence);
fclose(fptr);
return 0;
}
```

o/p:-

Enter a Sentence

I am awesom & so are files

pgm Explanation:-

after termination of this program, you can see a text file program.txt created in the same location where this program is located.

If you open and see the content, you can see the sentence : I am awesome & so are files.

In this program, a file is opened using opening mode "w".

In this mode, if the file exists, its contents are overwritten and if the file does not exist, it is created.

Then, the user is asked to enter a sentence. This sentence will be stored in file program.txt using fprintf() function.

(5) C Program to Read a Line from a file and Display it.

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
  char c[1000];
  FILE *fpt;
if ((fpt = fopen("program.txt",
          "r")) == NULL)
{
   printf("Error");
    exit(1)
}
 //reads text line until newline
fscanf(fpt, "%[^\n]", c);

printf ("Data from file:\n%s",
                    c);
fclose(fpt);
return 0;
}
```

Pgm Explanation

→ If the file program.txt is not found, this pgm prints error message.

=> If the file is found, the pgm saves the content of the file to a string c until '\n' newline is encountered.

Suppose, the program.txt file contains following text

C programming is nice
How are you doing?

The o/p of the program will be:

Data from file : C programming is nice

(6) C Program to display its own SOURCE Code

```
#include<stdio.h>
int main()
{
  FILE *fp;
  int c;
  fp = fopen(__FILE__,"r");
  do{
      c=getc(fp);
      putchar(c);
    }
  while(c!=EOF);
  fclose(fp);
  return 0;
}
```

=> Write and execute C program to find the average of numbers stored in sequential access file.

```
#include<stdio.h>
int main()
{
  int n1,n2,n3,n4,sum,gsize;
  FILE *infile;
  char fname[30];

printf(" Enter file name:");
gets(fname);

infile=fopen(fname,"r");

if(infile==NULL)
{
 printf("\n Can't Open");
 exit(1);
}

while(fscanf(infile,"%d %d
      %d %d %d",&gsize,
       &n1,&n2,&n3,&n4);
{
 printf(" GroupSize=%d \n
          n1=%d n2=%d
          n3=%d  n4=%d",
         gsize,n1,n2,n3,n4);
}

fclose(infile);
}
```

Count number of Characters in the File.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
   char ch;
   int count=0;
   FILE *fptr;

fptr=fopen("text.txt",
                "w");

if(fptr==NULL)
{
   printf("File can't open");
   exit(1);
}

printf("Enter some text");
while((ch=getch())!='\r');
{
    fputc(ch,fptr);
}
fclose(fptr);

fptr=fopen("text.txt","r");
printf("content of file");
while((ch=fgetc(fptr))!=EOF)
{
  count++;
  printf("%c",ch);
}
fclose(fptr);
Printf("No of character:%d",count);
getch();
}
```

Copy content of one file into another file

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
   FILE *f1, *f2;
   char ch;

 f1 = fopen("Sample.txt", "r");

 f2 = fopen("output.txt", "w");

 while(1)
 {
    ch = fgetc(f1);

    if(ch == EOF)
        break;
    else
       putc(ch, f2);
  }

printf("File Copied Successfully");

 fclose(f1);
 fclose(f2);
}
```

⇒Program to List all files and Sub-directories in a directory.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<dirent.h>

Void ListFiles(const char
                        *path);
int main()
{
  char path[100];
  printf("Enter path to list
            files");
  scanf("%s", path);

  ListFiles(path);
  return 0;
}
Void ListFiles(const char
 {                      *path)
   struct dirent *dp;
   DIR *dir=opendir(path);

   if(!dir)
        return;

 while((dp = readdir(dir))!=
 {                        NULL)

 printf("%s\n", dp->d_name);
 }
 closedir(dir);
}
```

Command line arguments

The most important function of C is main() function.

It is mostly defined with a return type of int and without parameters.

```
int main() { /* .... */}
```

⇒ We can also give command-line arguments in a C.

⇒ Command-line arguments are given after the name of the pgm in command-line shell of operating system.

⇒ To pass command line arguments, we typically define main() with two arguments:

↳ First argument is the number of command line arguments.

↳ Second is the list of command-line arguments.

```
int main(int argc, char *argv[])
                    { /* ... */}
```

or

```
int main(int argc, char **argv){/*...*/}
```

argc (ARGument Count)

↳ is int and stores no of command-line arguments passed by the user including the name of the pgm.

↳ So if we pass a value to a pgm, value of argc would be 2 (one for argument and one for pgm name)

↳ The value of argc should be non negative.

argv (ARGument Vector)

↳ is array of character pointers listing all the arguments.

↳ If argc is greater than Zero, the array elements from argv[0] to argv[argc-1] will contain pointer to strings.

→ Argv[0] is the name of the pgm, After that till argv[argc-1] every element is command-line argument.

Properties of Command Line Arguments:

(1) They are passed to main() function.

(2) They are parameters / arguments supplied to the pgm when it is invoked.

(3) They are used to control pgm from outside instead of hard coding those values inside the code.

4. argv[argc] is a NULL pointer

5. argv[0] holds the name of the pgm.

6. argv[1] points to the first command line argument and argv[n] point last argument.

C program to illustrate command line arguments.

```
#include<stdio.h>
int main(int argc, char* argv[]
{
 int counter;
printf("program Name Is: %s",
                    argv[0]);
if (argc==1)
  printf("In No Extra Command
          Line Argument
       Passed Other than
        Pgm Name");
if (argc >=2)
{
```