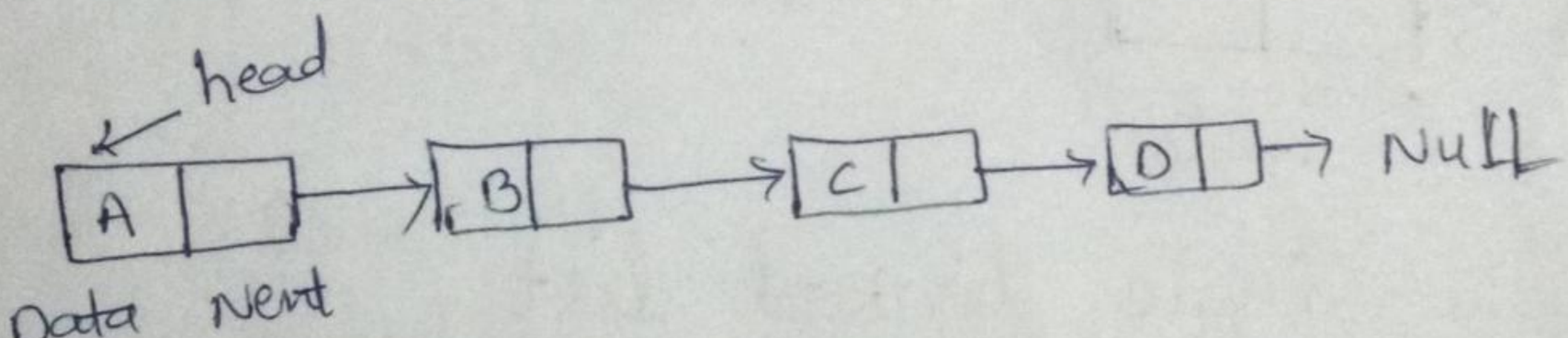
UNIT-V

Linked List Data structure

→ A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

→ In a linked list, The elements are linked using pointers.

In simple words, a linked list consists of nodes where each node contains a datafield and a reference (link) to the next node in the list.



Types of Linked list :-

There are 3 types of linked list

→ singly linked list

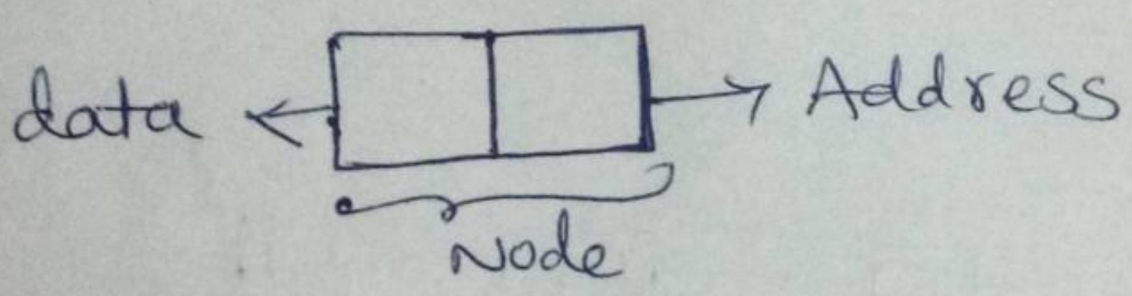
→ Double linked list

→ circular linked list.

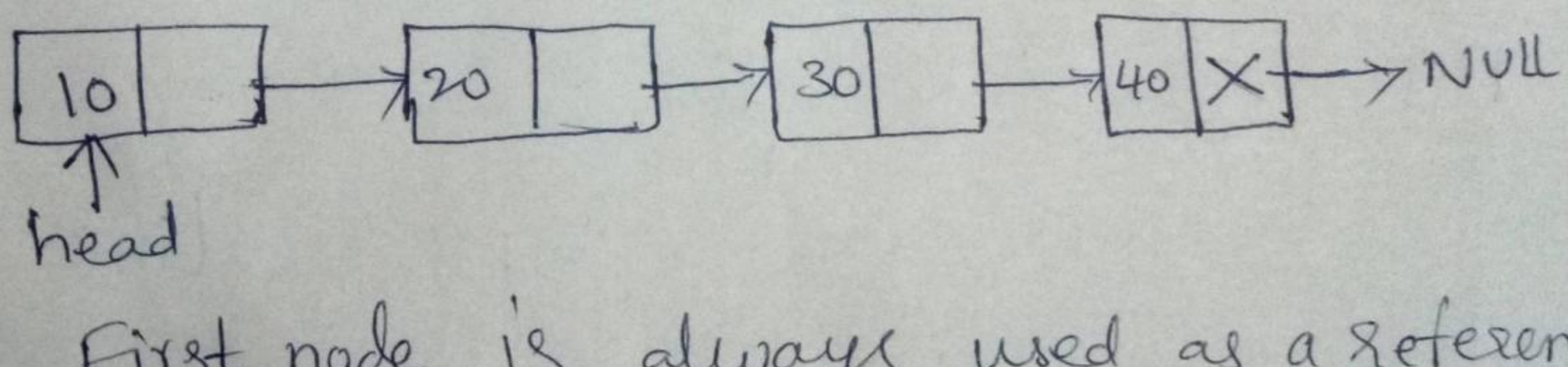
single linked list :—

single linked list is a basic linked list type. single linked list is a collection of nodes linked together in a sequential way where each node of single linked list contains a data field and an address field which contains the address (reference) of the next node.

Node



single linked list



→ The first node is always used as a reference to traverse the list is called HEAD (or) First

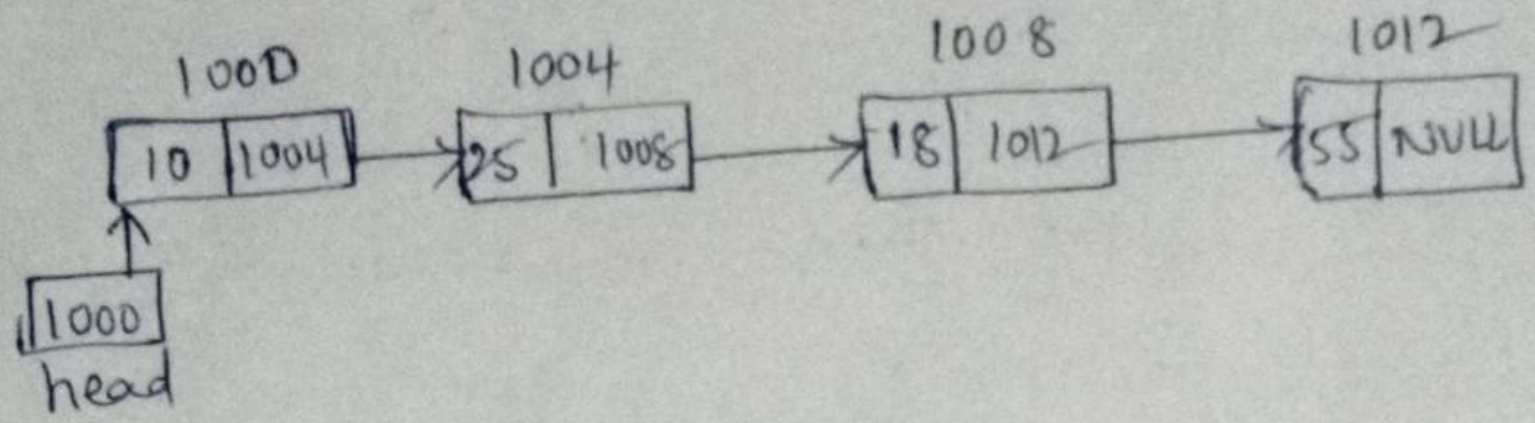
→ The last node points to NULL.

Node is represented as :

→ A Node can be defined as follows

```
struct node
  {
    int data;
    struct node * link;
  };
```

Ex:



operations on single linked list :—

The following operations are performed on a single linked list

① insertion
② Deletion
③ Display.

1. Insertion :— In a single linked list, the insertion opeeation can be performed in three ways. They are as follows

1. inserting at begining of the list
2. inserting at end of the list
3. inserting at specific location in the list.

1) Inserting at begining of the list :-

We can use the following steps to insert a new node at begining of the single linked list.

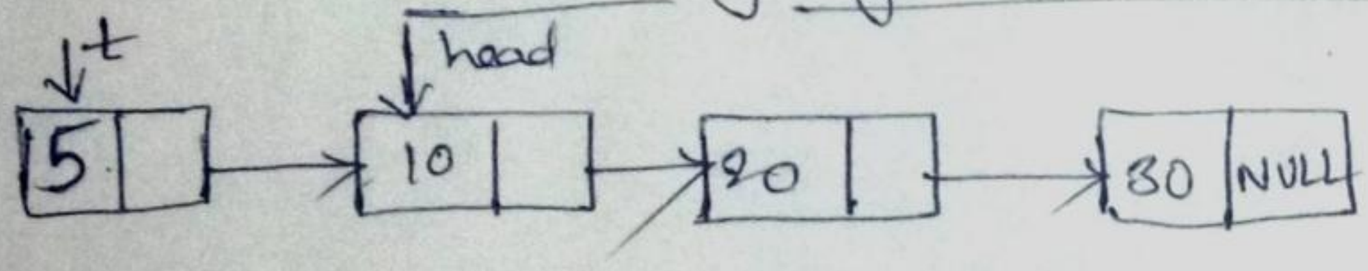
step 1 :- create a new node with given value

step 2: check whether list is empty (head == NULL)

step 3: if it is empty then, set newnode → link = NULL and head = newnode.

step 4: if it is Not empty then, set newnode → link = head and head = newnode.

Write a function to insert a node at the begining of the linked list




```
struct node *insertfirst (x, head)
        int x;
    struct node * head;
    {
        struct node *t;
    t = (struct node *)malloc( sizeof (structnode));
        t → data = x;
        t → link = head;
        head = t;
        return (head);
    }
```

2) Deletion :-

In a single linked list, the deletion operation can be performed in three ways. They are as follows

1. Deleting from begining of the list
2. Deleting from end of the list
3. Deleting a specific Node.

1 Deleting from begining of the list :-

We can use the following steps to delete a node from begining of the single linked list.

Step 1: check whether list is empty (head == NULL)

Step 2: if it is empty then, display 'list is empty' deletion is not possible. and terminate the function.

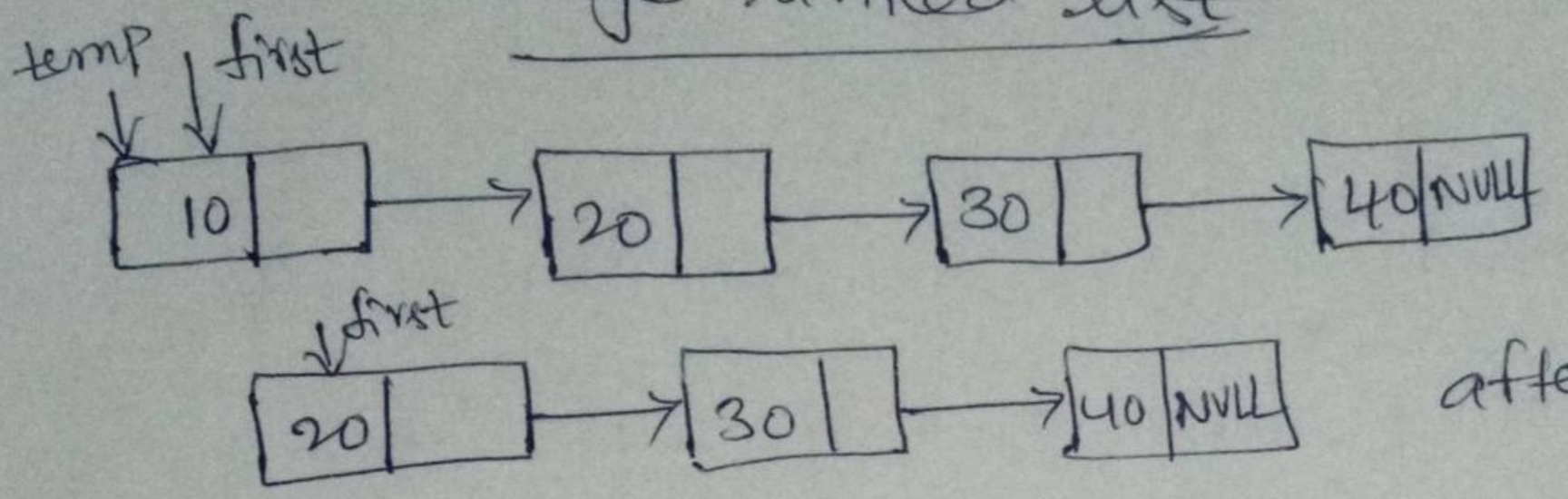
Step 3: if it is not empty then, define a node pointer temp and initialize with head.

Step 4: check whether list is having only one node (temp → next == NULL) next (or) link

Step 5: if it is true then set head = NULL and delete temp

Step 6: if it is false then set head = temp → next and delete temp.

Write a function to delete a first node in single linked list



before deletion

after deletion.

```
struct node * deletefirst ( head, x)
    int x;
    struct node * head;
    {
      struct node *t;
    if( head == NULL)
    {  return (NULL);
    }
else
{   t = head;
    head = head → link
    x = t → data;
    free (t);
    return ( head);
}
}.
```

3) Displaying a single linked list :-

We can use the following steps to display the elements of a single linked list.

step 1: check whether list is empty (head == NULL)

step 2: if it is empty then, display List is Empty and terminate the function.

step 3: if it is not empty then, define a node pointer temp and initialize with head

step 4: Keep displaying temp → data with an arrow (→) until temp reaches to the last node

step 5: Finally display temp → data with arrow pointing to NULL (temp → data → NULL)

Write a function to display the S.L.L :-

```
void display( head)
    struct node *head;
{
    struct node *t;
    if( head == NULL)
printf(" linked list is Empty\n");
else
{   t = head;
    while( t != NULL)
```

```
{
printf(" %d \t %u \n", t->data, t->link);
    t = t->link;
}
printf("\n");
}
```

Advantages of linked list :—

1. linked list is dynamic data structure
2. linked list (size) can grow and shrink during run time (Dynamic size)
3. insertion and deletion operations are easier
4. Efficient memory utilization (no wastage of memory)
5. Linear data structures such as stack, queue can be easily implemented using linked list.

Linked list and Arrays are both used for storage of elements, but both are different techniques. In an array elements are one after the another (successive memory allocation), but in linked list, memory is not contiguous.

Difference between sequential list (arrays) and linked list

Array	Linked list
1) Array is a collection of elements of similar data.	1) It is an ordered collection of elements of same type, which are connected to each other using pointers.
2) Array supports random access. which means elements can be accessed directly using their index like arr[0], arr[6] etc.	2) Linked list supports Sequential Access. which means to access any element/node in a linked list.
3) In an array elements are stored in contiguous memory allocation in the memory.	3) In a linked list new elements can be stored anywhere in the memory.
4. Before using array we have to specify the size of array	4. Before using S.L.L we need not specify the linked list size

Program on singk linked list

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

    struct node
        {
        int data;
        Struct node *link;
        };
Struct node *slcreate()  // function to create
                                S.L.L
    {
    struct node * head, *prev, *pres, *temp;
        int x;
    head = (struct node *) malloc(sizeof(struct node));
        prev = head;
printf("enter the data of the nodes terminated
                        by ^D enter key \n");
        while(scanf("%d", &x) != EOF)
        {
pres = (struct node*) malloc(sizeof(struct node));
pres -> data = x;
    prev -> link = pres;
        Prev = pres;
    }
```

```
pres -> link = NULL
    temp = head;
    head = head -> link;
        free(temp);
        return(head);
}
void main()
    {
        int x, ch;
    struct node *head, *insertfirst(), *deletefirst(),
                    *slcreate();
        void display();
    head = slcreate();
printf("single linked list operations\n");
    do
    {
        printf("1. insertion\n");
        printf("2. deletion\n");
        printf("3. display\n");
        printf("4. exit\n");
    printf("enter your choice\n");
        scanf("%d", &ch);
    switch(ch)
        {
```

```
case 1:
    {
        printf("enter the data \n");
        scanf("%d", &x);
      head = insertfirst(x, head);
      break;
    }
case 2:
    {
      head = deletefirst(&x, head);
      if(x==-1)
      printf("linked list is empty
                    deletion is not possible\n");
      else
      printf("the data deleted from the node
                    is %d\n", x);
      break;
    }
case 3:
      display(head);
      break;
case 4:
      printf("program ends\n");
      break;
default:
      printf("wrong choice\n");
      break;
}
```

```
} while (ch!=4);
}
struct node * insertfirst( x, head)
        int x;
        Struct node * head;
    {
      struct node *t;
    t = (struct node *) malloc(sizeof(struct node));
      t -> data = x;
      t -> link = head;
        head = t;
     return(head);
    }
struct node * deletefirst(x, head)
        int *x;
      struct node * head;
      {
        struct node *t;
        if( head == NULL)
      {
          * X = -1;
          return (NULL);
      }
    else
      { t = head;
      head = head -> link;
      * x = t -> data;
```

```
        free(t);
        return(head);
      }
    }
void display(head)
    struct node *head;
    {
      struct node *t;
    if(head == NULL)
    printf("linked list is empty\n");
else
    {
      t = head;
      while(t != NULL)
       {
      printf("%d\t%u ", t->data, t->link);
        t = t->link;
      }
      printf("\n");
    }
}.
```

File management in C

→ A file can be used to store a large volume of persistent data.

(or)

→ A collection of data which is stored on a secondary device like a hard disk is known as a file.

A file is generally used as real-life applications that contain a large amount of data.

File operations :—

C language provides following file management functions (or) operations

1. Creating (or) creation of a file
2. opening a file
3. Reading a file
4. writing to a file
5. closing a file.

The library functions which are used for operating the files are:

1. High-level file I/O functions
2. Low-level file I/O functions

Following are the most important file handling (or) file management functions available in 'c'.

High level I/O Functions

Function Name	Purpose
fopen()	creating a file (or) opening an existing file
fclose()	closing a file
fprintf()	writing a block of data to a file
fscanf()	Reading a block data from a file
getc()	Reads a single character from a file
putc()	Writes a single character to a file
getw()	reads an integer from a file
putw()	writing an integer to a file
fseek()	Sets the position of a file pointer to a specified location
ftell()	Returns the current position of a file pointer
rewind()	Sets the pointer at the beginning of a file.

How to create and open a file :-

→ whenever you want to work with a file, the first step is to create a file.

To create a file in a 'c' program

Syntax : FILE *fp;

To open a file :-

Syntax : fp = fopen("file name", "mode");

→ In the above syntax, the FILE is a data structure which is defined in the standard library. (stdio.h)

→ fopen is a standard function which is used to open a file to perform operations

- if the file is not present on the system then it is created and then opened
- if a file is already present on the system, then it is directly opened using this function.

→ fp is a file pointer which points to the type file.

→ fopen is a function which opens a file in the specified mode. They are six modes

1) r 2) w 3) a
4) r+ 5) w+ 6) a+

1) r (read) :- open the file for reading only

```
fp = fopen("abc.txt", "r");
```

fopen function opens the file in read mode and also fp is the pointer to the file abc.txt. if the file is not existing then fopen() function returns NULL. It is stored in fp.

```
if( fp == NULL)
{
printf("sorry The file is not existing\n");
exit(0);
}
```

→ we can only read but we can't write new data in to the file.

2) w (write) :- open the file for writing only.

```
fp = fopen("abc.txt", "w");
```

The file is opened in write mode.
whenever a file opened in "w" mode means it should be a non existing file. in such case fopen creates a new file and fp points to it

→ we can only write into the file and we can't read data from the file.

3) a (append) :- open the file for appending (adding) data to it.

```
fp = fopen("abc.txt", "a");
```

The file is opened in append mode.

The advantage of append mode is the existing contents of file are not lossed and new data is appended to the end of the file.

→ when a file is opened in append mode we can only write in to the file but we can't read from the file.

4) r+ :- The existing file is opened to the begining for both reading & writing.

```
fp = fopen("abc.txt", "r+");
```

we can peeform 3 operations on file

(i) read the existing contents of file
(ii) append new data to the file
(iii) alter existing contents of file.

5) w+ :- same as w except both for reading & writing.

(i) read (ii) append (iii) alter

only non existing files should opened in "w+" mode if a file exists and opened in w+ mode then the existing contents of the file will be losed.

6) a+ :- same as a except both for reading & writing.

(i) read (ii) append.

we can't alter the existing contents of file. only existing files should be opened in a+ mode. if a file not existing and opened in a+ mode it returns NULL

closing a file:—

A file must be closed as soon as all operations are completed.

→ fclose() function closes the file the is being pointed by file pointer fp.

FILE *P1, *P2;

Syntax: fclose(fp);

Ex: fclose(P1), fclose(P2)

File I/O Functions:—

once a file is opened, reading (or) writing data to / from a file. using the standard I/O functions.

Reading data from a file:—

The simplest file input functions are:

1) getc() 2) fgets() 3) getw() 4) fscanf()

① getc():—

The getc() function is used to read a single character from a file that has been opened in read (r) mode.

Syntax: character variable = getc(fp);

The file pointer moves by one character position for every operation of getc().

The getc() will return an end-of-file marker

EOF, when end of the file has been reached

2) getw() :-

→ The getw() and putw() are integer-oriented functions. They are similar to the getc() and putc() functions.

→ These functions is used to read and an integer value from a file.

Syntax :- Variable = getw(file pointer);

ex : n = getw(fp);

3) fgets() :-

→ fgets() function is a file handling function in c language.

→ which is used to read a file line by line.

Syntax :- fgets(string, n, fp);

→ string is the variable in which the string is stored

→ n is the maximum length of the string that should be read

4) fscanf() :-

→ It is used to read formatted data from a file.

Syntax : fscanf(fp, "control string", list);

The list may include variables, constants and strings

Ex: fscanf(fp, "%d", &age);

fscanf(" fp, "%f", 7.5);

Writing data into a file :—

The simplest file output functions are:

① putc(c) 2) putw(c) 3) fputs() 4) fprintf()

① putc(c) :—

This function is used to writes a single character to a file.

Syntax :— `putc(c, fp);`

writes the character contained in the characte variable c to the file associated with FILE Pointer fp.

2) putw(c) :—

This function is used to write integer value to a file.

Syntax :— `putw(integer, fp);`

3) fputs(c) :—

This function is used to write(print) a string to the file.

Syntax :— `fputs(string, fp);`

Ex: fputs(str, fp);

str is the name of string variable.

4) fprintf() :-

fprintf() is same as printf() function but instead of writing data to the console, it writes formatted data into the file.

Syntax :- `fprintf(fp, "controlstring", list);`

ex : `fprintf(fp, "%d", age);`

1) W.C.P to read data from console & write data into File (read char from a file)

```
#include<stdio.h>
void main()
{
FILE *fp;
    char ch;
    fp = fopen("input.txt", "w");
while((ch= getchar())!=EOF)// reading data from console
{
fputc(ch, fp);   // writing data into file
// writing data on the screen
}
fclose(fp);
}
```

o/p:

Input.txt

hai welcome

hai welcome

enter ctrl + d

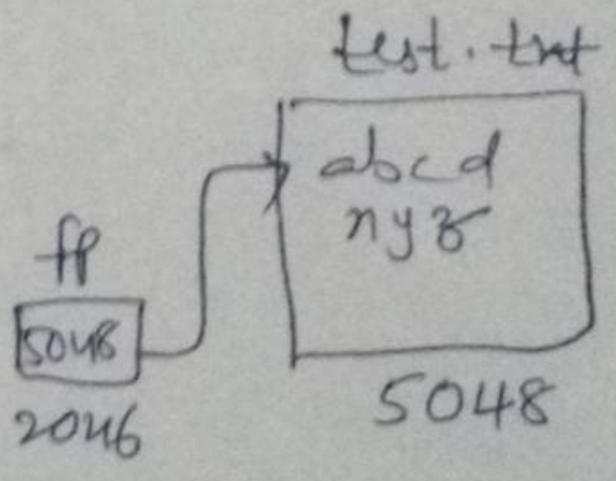
EOF represented by ctrl + d

2) program to Read data from file & write data on screen.

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp = fopen("input.txt", "r");
while((ch=fgetc(fp)) != EOF) // reading data from file
{
printf("%c", ch); // writing data on the screen.
}
fclose(fp);
}
```

3) w.c.p to check whether the file is present or not

```
#include<stdio.h>
void main()
{
FILE *fp;
fp = fopen("c:/examples/test.txt", "r");
if( fp == NULL)
printf("file not present\n");
else
printf("file opened in readmode\n");
}
```



4) w.c.p to copy the information of one file into another file.

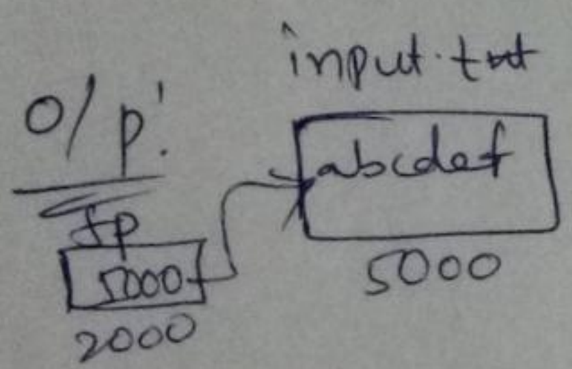
```
#include<stdio.h>
void main()
{
FILE *fp1, *fp2;
    char ch;
fp1 = fopen("d:/input.txt", "r");
fp2 = fopen("d:/output.txt", "w");
while((ch=fgetc(fp1)) != EOF)
    {
        fputc(ch, fp2);
    }
printf("copied");
    fclose(fp1);
    fclose(fp2);
}
```

5) w.c.p to write and read data from a file.

```
#include<stdio.h>
void main()
{
    FILE *fp1, *fp2;
    char ch;
fp1 = fopen("input.txt", "w");
printf("enter data\n");
```

```
while((ch=getchar())!=EOF)// reading data from the keyboard
    {
    fputc(ch, fp1); // writing data into file
     }
   fclose(fp1);
 fp2 = fopen("input.txt", "r");
 printf("file data\n");
 while((ch = fgetc(fp2))!=EOF)// reading data from file
    {
       printf("%c", ch); // writing data on the screen
    }
 fclose(fp2);
 }.
```

enter data
abcdef.

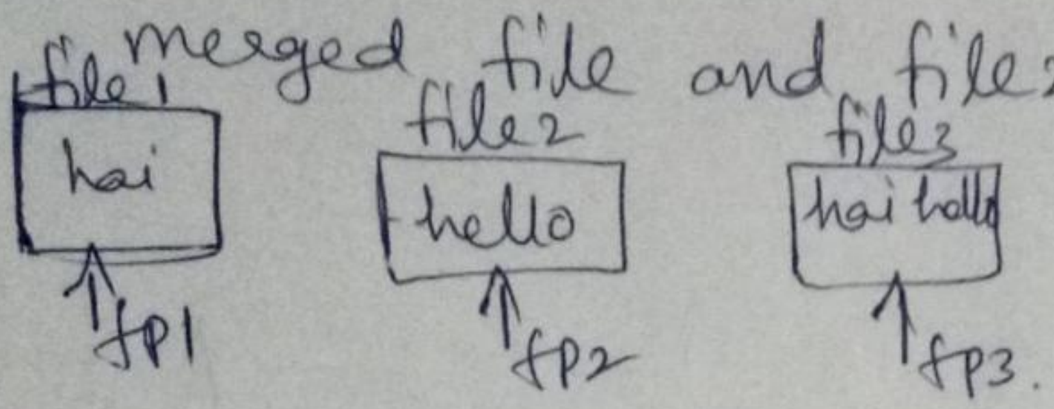


6) W.C.P to merge two files contents into a third file.

```
#include<stdio.h>
 void main()
   {
  FILE *fp1, *fp2, *fp3;
      char ch;
  fp1 = fopen("file1", "r");
  fp2 = fopen("file2", "r");
  fp3 = fopen("file3", "w");
while((ch = fgetc(fp1))!=EOF)
   fputc(ch, fp3);
while((ch = fgetc(fp2))!=EOF)
   fputc(ch, fp3);
```

```
printf(" merged file1 and file2");
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
}.
```

o/p:



Random access File Functions :-

There is no need to read each record sequentially, if we want to access a particular record.

C support these functions for random access file Processing.

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

1) fseek() :-

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

Syntax :- fseek(filepointer, displacement, pointer position);

here displacement - it is positive (or) negative. This is the number of bytes which are skipped backward (if negative) (or) forward (if positive) from the current position.

Pointer Position:

This sets the pointer position in the file

Value	Pointer position
0	beginning of file
1	Current position
2	End of file

Ex: fseek(fp, 10L, 0);

0 means pointer position is on begining of the file, from this statement Pointer position is skipped 10 bytes from the begining of the file.

Ex: fseek(p, -5L, 1);

from this statement pointer position is skipped 5 bytes backward from the current positn

2) ftell() :-

This function returns the value of the current position in the file. The value is count from the begining of the file.

Syntax: ftell(fptr);

where fptr is a file pointer

3) rewind() :-

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

Syntax: rewind(fptr);

where fptr is a file pointer.

example program for fseek():

1) write a program to read last 'n' characters of the file.

```
#include<stdio.h>
Void main()
{
   FILE *fp;
    char ch;
fp = fopen("file1.c", "r");
   if (fp == NULL)
   printf("file can't be opened");
else
{
   printf("enter value of n to read
           last 'n' characters");

   scanf("%d", &n);
   fseek( fp, -n, 2);
while(( ch= fgetc(fp)) != EOF)
   {
      printf("%c\t", ch);
   }
}
fclose(fp);
}.
```

command Line Arguments in C

→ The arguments passed from command line are called command line arguments.

→ These arguments are handled by main() function.

→ command line arguments are passed to the main() method.

Syntax : int main(int argc, char *argv[])

→ here argc counts the number of arguments on the command line

argv[] is a pointer array which holds pointers of type char which points to the arguments passed to the program.

Example program for commad Line Arguments

```
#include <stdio.h>
int main( int argc, char *argv[])
{
  int i;
  if( argc >=2)
  {
    printf(" The arguments supplied are: \n");
    for( i=1; i<argc; i++)
    {
      printf(" -%s\t", argv[i]);
    }
  }
  else
    printf(" argument list is empty \n");
}
```