PAGE EDGE
Date

* Bit field

Bit field is mechanisum where we can allocate memory in the form of bits.

Syntax :-

```
datatype variablename : no_of_bits ;
```

⇒ bitfield variable must and should be declared within structure or union

⇒ make bit field as unsigned type so all the bits act as data bits

⇒ float type bit field is not possible

⇒ there is no any specific format specifier is available for bit field

⇒ address operator (&) is not aplicable on bit field.

⇒ sizeof operator is not aplicable on bit field.

EX :-

```
#include<stdio.h>
struct st
{
    int i;
    unsigned int j:3;
};
void main()
{
    struct st v;
    v.j = 7;
    scanf("%d", &v.j);  → invalid
    pf("%d\n", v.j);
    pf("%ld\n", sizeof(v.j));  → invalid
}
```

* sizeof operator is not applicable on bit field than How to find sizeof bit field ?

```
#include<stdio.h>
struct st
{
    int i;
    unsigned int j : 4;
};
void main()
{
    ↳ logic to find size of bitfield
```

```
struct st v;
v.j = 1;
int c=0;

while (v.j)
{
   v.j = v.j << 1;
   c++;
}

printf("sizeof v.j : %d bits \n", c);
}
```

* Print data of struct. using another function

```
#include<stdio.h>
struct st
{
  int rollno;
  char name[20];
  float marks;
};

void print(int, char *, float);
void print1(struct st);
void print2(struct st *);

void main()
{
  struct st v={5, "abc", 15.5};
  print(v.rollno, v.name, v.marks);
  print1(v);
  print2(&v);

  printf("from main : %d %s %f \n",
        p->rollno, p->name, p->marks);
        v.rollno, v.name, v.marks);
} p

void print2(struct st *p)
{
  pf("from print2 : ...", p->rollno, p->name, p->marks)
  p->rollno = 10; ← actual data - change
}

void print1(struct st v)
{
  pf("from print1 : ...", v.rollno, v.name, v.marks);
  v.rollno = 10; ← actual data - No change
}

void print(int r, char *n, float m)
{
  pf("from print : %d %s %f", r, n, m);
}
```

* WAP to design a function which will return 3 diff. types of data which are int, string and float.

```
#include<stdio.h>
struct st
{ int rollno;
  char name[20];
  float marks; };
```

```
struct st return_ist(void);
void main()
{
    struct st v = return_ist();
    pf("%d %s %f \n", v.rollno, v.name, v.marks);
}
struct st return_ist(void)
{
    struct st v={ 5, "abcd", 45.5 };

    return v;
}
```

* WAP to design a function which allocate dynamic memory for one studed database and return starting adrs of that.

```
#include <stdio.h>
#include <stdlib.h>
struct st
{
    int rollno;
    char name[20];
    float marks;
};
struct st * ret_dma(void);

void main()
{
    struct st *p;
    p = ret_dma();

    pf and sf => &p->rollno, p->name, &p->marks.
```

PAGE EDGE

Date :

```
    pf("%d %s %f", p->rollno, ...);
}

struct st * ret_dma(void)
{
    struct st *p = malloc(sizeof(struct st));

    return p;
}
```

* WAP to design function which allocates dynamic memory for one student database and return starting adrs of that but return type of function is void

```
#--
#---
struct st
{
    int rollno;
    char name[20];
    float marks;
};
void ret_dma(struct st **p);
void main()
{
struct st *p;
ret_dma(&p);

pf & sf => &p->rollno,...;
pf => &p->rollno,...
}
```

```
void ret_dma(struct st **p)
{
*p = malloc(sizeof(struct st));
}
```

offset → distance b/w two memory ~~of~~ content

PAGE EDG3
Date .....

* Structure Padding

⇒ Allocation of sum extra bytes of memory for structure variable than the required one it's called as a structure padding.

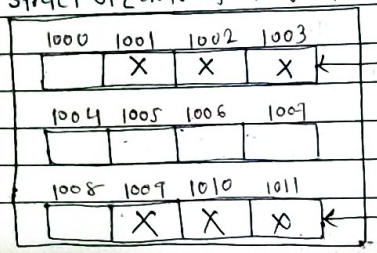
→ Structure padding is compiler dependant process

→ if processor is 8 bit than it can fetch 1 byte at a time
if processor is 32 bit than it can fetch 4 byte at a time

→ Based on this when memory is allocated for 4 or more than 4 bytes compiler request to operating system allocate memory in the multiple of 4 bytes

→ To increase the speed of execution and processing -when adrss are in multiple of 4s offsets no need to calc. every time. This improve speed.

struct st {char a, int b, char c} s;



in above example char a; takes 4 bytes int b; takes 4 bytes and char c; also take 4 bytes.
Here 3 bytes from a and 3 bytes from c are unused or 4 can say wasted.
such unused memory locations in structure due to padding are called as **holes** in a structure

* How to avoid structure padding ?

option ① : Place similar types of data together to avoid structure padding (~~not~~ to avoid memory wasted)

How to pack structure ?
what is structure packing ?
sometime it may required to avoid memory wasted which is nothing but extra bytes of memory allocated b/w two member location its called as **structure packing**

These are 2 ways to make structure paking

op ① **# Pragma pack (1)**

op ② when structure declaration is going to end at that time u need to mention
--attribute-- ((packed));

Structure Padding

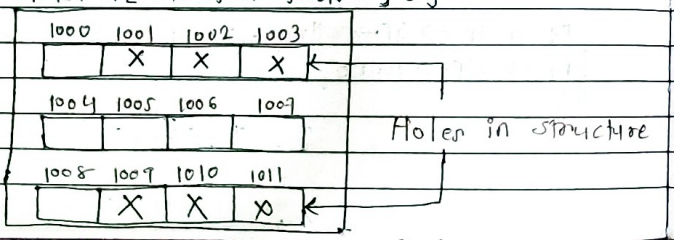
⇒ Allocation of sum extra bytes of memory for structure variable than the required one its called as a structure padding.

→ Structure padding is compiler dependant process

→ if processor is 8 bit than it can fetch 1 byte at a time
if processor is 32 bit than it can fetch 4 byte at a time

→ Based on this when memory is allocated for 4 or more than 4 bytes compiler request to operating system allocate memory in the multiple of 4 bytes

→ To increase the speed of execution and processing -when adrss are in multiple of 4s offsets no need to calc. every time. This improve speeds



offset → distance b/w two memory ~~of~~ element

PAGE EDGE
Date

in above example char a; takes 4 bytes int b; takes 4 bytes and char c; also take 4 bytes
Here 3 bytes from a and 3 byte from c are unused. or 4 can say wasted.
such unused memory locations in structure due to padding are called as **holes** in a structure

* How to avoid structure padding ?

option ① : Place similar types of data together to avoid structure padding (~~not~~ to avoid memory wasted)

How to pack structure ?
what is structure packing ?
sometime it may required to avoid memory wasted which is nothing but extra byte of memory allocated b/w two member location its called as **structure packing**

There are 2 ways to make structure paking

op ① #pragma pack (1)

op ② when structure declaration is going to end at that time we need to mention
__attribute__ ((packed));

memory managment and speed of process both are inversly proposnal

```
Ex:  struct one
     {
       char i;
       char i1;
       char i2;
       int j;
       int j1;
       int j2;
     } __attribute__((packed));
```

Union

union is one of the usedefine datatype which is collection of diff. typer of data which are stored in same memory location.

```
#include<stdio.h>

union u
{
  int i;
  char ch;
  float f;
};

void main()
{
```

~~char d[5]~~ ⇒ size :- ~~5 byte~~ ⇒ ~~allocated~~ mem : ~~8 bytes~~

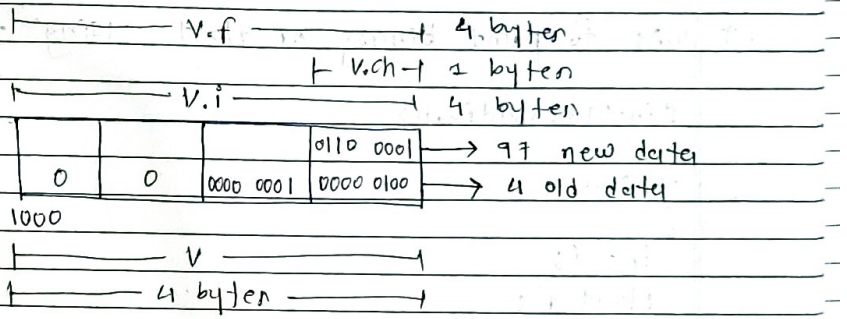
PAGE EDGE

Date :

```
union u v;

pf("%u \n", &v.i);  → 1000
   "     &v.ch → 1000
   "     &v.f  → 1000
}
```

union variables memory allocation happens based on a largest member size.



Ex :

```
v.i = 260;
pf("%d \n", v.ch);  → 4
v.ch = 'a';
pf("%d \n", v.i);   → 353
```

* WAP to check given system is little endian or big endian using union data type.

```
#---
union u
{
  int i;
  char ch;
};
```

```
void main()
{
 union u V;
  v.i = 10;
  if (v.ch == 10)
      pf ("little endian \n");
  else
      pf ("Big endian \n");
}
```

* WAP to print binary of float using union

```
#include<stdio.h>

union u
{
 int i;
 float f;
};

void main()
{
  union u v;
  v.f = 23.5;
  int pos;

for ( pos = 31 ; pos >= 0 ; pos-- )
      pf ("%d", v.i >> pos & 1 );
pf ("\n");

}
```

PAGE EDGE

Date .....

* union inside struct

```
struct a
{
      union b
      {
          int i;
          char ch[3];
      } B;  size of B → 4
      union c
      {
          float f;
          int *p;
      } c;  size of c → 8
} A;
   size of A → 12.
```

→ B and c are members of structure

→ How to access i ?
⇒ A.B.i

* struct and union insid union (nested union)

```
union a
{
    struct b
    {
       int i;
       char ch[3];
    } B;  size of B → 7
    union c
    {
       float f;
       int *p;
    } c;  size of c → 8
} A;
   └→ size of A → 8
```

typedef storage_classer int aa; → X

struct rrq	struct wrq	struct ack	struct data	St. error
{	{	{	{	{
};	};	};	};	};

union tftp
{
  struct rrq r;
  struct wrq w;
  struct ack a;
  struct data d;
  struct error e;
}

Typedef

Typedef is one of the user defined datatype which is used to provide another name to existing datatype.

so that understanding of datatype or variable becomes easy.

Syntax :

```
typedef old_datatype_name  new_name ;
```

new_name should not be any keyword.

typedef int aaa;
⇒ sign_type aaa i; → X
⇒ size_type aaa i; → X
type_qualifier aaa i; → ✓
storage_classer aaa → ✓

① EX:

```
#include<stdio.h>

typedef int INTEGER ;

void main()
{
  INTEGER i ;
  pf("%d \n", sizeof(i)); → 4
}
```

② EX: (signed type)

```
//typedef char 58 ;

typedef unsigned char u8 ;

void main()
{
  unsigned 58 ch ; → invalid (we cant provide signe type to new name of data type)
  u8 ch;
  ch = 259 ;
  pf("%d \n", ch); → 1
}
```

③ EX: (pointer)

int *p ; → p is variable of int * ;

typedef int *p ; → p is another name to int *.

#---

typedef int * P;

void main()
{
  P q, *r; → P is datatype and q is variable of it.
           → r act as int double pointer.
  int i=20;
  q = &i;
  pf("%d \n", *r);

④ EX: (array)

#---

//int q[5]; → q is variable of int array

typedef int q[5]; → q act as another name to int array

void main()
{
  q b, p[3]; → q is datatype and b is variable of it
             → p is 2d array
             → in this 2d array 3 rows and 5 cols
  int i, r;

  r = sizeof(p) / sizeof(p[0]);

  pf("%d \n", r); → 3

for(pf and sf → ... &b[i] ;)
for(pf , → ... b[i])&

}

we can give multiple name to one datatype.
ex: ↓

PAGE EDGE
Date

⑤ EX: (struct)

#---

typedef struct one
{
  char i;
  int j;
} V; → V act as another name to struct one datatype

typedef struct one y; → y is another name to struct one.

void main()
{
  V a = {'a', 10}; → V is datatype and a is variable of it.
  pf("%c %d", a.i, a.j);
}

* Application or one of the benifits.

#---

① typedef int my_int;

void f1(my_int i, my_int j)
{ }
void f2(") { }

```
void main()
{
  my_int i,j;
  f1(i,j);
  f2(i,j);
}
```

if we want to work with char, then we need to change only in line no. ①.

enum

→ enum is one of the user defined datatype which is used to provide name to int constants.

so that understanding of program becomes easy.

EX:

#---

enum day { mon, tue, wed, thu, fri, sat, sun };

Pf → "%d", mon → 0
"%d", wed → 2

~~enum day~~ mon++ ; invalid
mon = 5 ; invalid

enum day { mon=5, tue, wed ... };

Pf → "%d", mon → 5
"%d", wed → 7

→ by default int ~~const~~ starts from 0.

→ enum day { mon, tue }; } invalid - error
enum day1 { mon };

→ fun (mon); → for this fun formal parameter should be int.

PAGE EDGE
Date :

→ enum day { mon = 5, tue, wed = 5, thu, fri, sat };
5 6 5 6 7 8

→ EX:
How to use enum:

#---
enum colour { red, green, blue };

void main()
{
int input;
Pf and sf → %d, &input.;

if (input == red)
Pf ("red \n");
else if (input == green)
Pf ("green \n");
else
Pf ("blue \n");
}

→ memory allocation

* Preprocessor

→ one of the compilation stage
→ input ⇒ .c (source file)
output ⇒ .i (pure c file or extenden file)
→ cmd: cc -E file.c -o file.i
→

include the header filen :
#include<>, #include" "

→ remove the comments :
// single line, /* multi line */

→ Replacement of macros :
→ predefined macros
→ user defined macros function
• without arg. (object type macros)
• with arg. (function type macros)

→ condition compilation :
the benifit of conditional compilation is to reduce the size of executable file

→ Preprocessor directives are special word which are useful for communicate with preprocessor (keyword)

→ All the preprocessor directives starts with #.

→ #include, #define, #if, #ifdef, #ifndef, #else, #elif, #endif, #prag, #undef.

PAGE EDGE
Date .....

* Header file inclusion :-

Q what is header file and what it contains ?
Header file is one type of file which contains all the types of declaration (statement for what memory is not allocated)

EX: variable declaration, function declaration, structure, union, typedef, enum, #define etc.

Q why header file in c progoarming ?
As per industry standard when we work on any project based task which contains multiple files in such situation instead of writing declaration in each and every file create one sperate file and write all the declaration in that file.

so, if any declaration problem occurs it will very easy to identify and to solve the problem.

Q is it mandatory to provide .h extention to header file ?
No, it is not mandatory.

Q then why all the predefine header file having .h extention ?
in multiple file base task which file is header file containing all the declaration to

identify .h extention become helpful bcoz of this reason all the predefine header file having .h extention

? what is the diff b/w #include <stdio.h> and #include "stdio.h" ?

#include < > : search in predefined path only

#include " " : search in present working directory if not present then after search in predefined path.

? what is header file inclusion by preprocesso ?
" is first search for a header file where it is available if it is found then it will copy header & contain into .i file

? what is meaning of comment removal by preprocessor ? it replace commented line portion with spaces in .i file

EX:
```
#include <stdio.h>
#include "header.h"

void main()
{
    // Printf("One\n");
    INTEGER num1 = 5, num2 = 10;
    /* one
       two */
    Pf("%d\n", sum(num1,num2));
}
```

exa.c

PAGE EDGE
Date

```
INTEGER sum (INTEGER num1, INTEGER num2)
{
    return num1 + num2;
}
```

header.h

```
typedef int INTEGER;
int sum (INTEGER, INTEGER);
```

* Replacement of macro

? what is macro replacement by pre processor. ?
macro replacement by pre processor is nothing but one small part of program which will replace with another part by pre processor

? what is macro ?
macro is nothing but one small statment which is present in a program.

syntax:

#define macroname macrobody

ex:-
#define PI 3.14

EX:-

macro names must be identifiers : means it follow variable creation rules but keywords are allowed.

EX ①

```
#include <stdio.h>

#define INTEGER int

void main()
{
  INTEGER i=10;
  printf("%d \n", sizeof(i));
}
```

EX ②

```
#include <stdio.h>
#define PF printf

void main()
{
  PF("hello \n");
}
```

EX ③

```
#include <stdio.h>
#define 1 ;   → invalid bcoz macro name is
                not identifier.
void main()
{
  pf("hello \n")1
}
```

[5] what is a diff. b/w macro without arg. and typedef ?
or
what is a diff. b/w #define and typedef ?

PAGE EDGE
Date

Marco without argument	Typedef
→ #define is preprocessor dir.	used defining datatype
→ #define macroname macbody	typedef oldnam newnam;
→ #define is useful for comm: unicatting with preprocessor	typedef is useful to comm. wit translator
→ is used for data types, symbol	only for data type
replace with any othe symbol	
→ Replacement takes place	No Repl. takes place.
→ #define IPTR int *	Typed int * INTPTR
IPTR P, q ;	INTPTR p, q
int *p, q	↳ Both are act as
↳ only P is acts as pointer	pointer

EX ④

```
#include <stdio.h>

#define int char
#define char float
#define float int.

void main()
{
  int i; → char i → float i → int i
  int ch; → float ch → int ch → char ch;
  float f; → int f → char f → float f;

  pf("%d %d %d", sizeof(i), sizeof(ch), sizeof(f));
                  ↓          ↓          ↓
                  4          1          4
```

each line (#define) execute only one time for one symbole (macro name)
order of #define doesn't matter

* Macro with Argument

EX ①
```
#include<stdio.h>
int sum(int a, int b)
{
  return a+b;
}
```
⇒
```
#define SUM(a,b) a+b

void main()
{
  // int i=2, j=3, r;
     float i=2.5, j=3.5;

  r = sum(i,j);
  pf("r = %d \n", r);

  r = SUM(i,j);
```
→ macro name replaced by macro body
⇒ `r = a+b;`
→ arg replacement takes place.
⇒ `r = i+j;`
```
  pf("r = %d \n", r);
}
```

* what is diff. b/w macro with arg. and function?

Macro with arg.	function
→ in the place of macro call macro body get pasted	in place of function call function body not get placed.
→ macro arg. doesn't have any type	function arg. having specific type.
→ in above ex. sum macro for adding two floats	we cannot use same sum fun. for adding two floats
→ macro are faster	function are slower.
→ if task is smaller define a macro	if task is bigger define function.

EX ②
```
#include<stdio.h>
int mul(int a, int b)
{
  return a*b;
}
① #define MUL(a,b) a*b
② #define MUL(a,b) (a)*(b)

void main()
{
  int i=2, j=3, r;
  r = mul(i+2, j+1);
  pf("r = %d \n", r);

  r = MUL(i+2, j+1);
```
① `r = a*b`
`= i+2*j+1`
`= 9`

⇒ ② `r = (a)*(b)`
⇒ `r = (i+2)*(j+1)`
`= 16`
```
  pf("r = %d \n", r);
}
```

* WAP to design a macro for swaping of two nums.

```
#include<stdio.h>
#define SWAP(a,b,type) { type t; \
                         t = a; \
                         a = b; \
                         b = t; }
```

in " " (string constant), preprocessor cannot perform any task (replacement)

```
void main()
{
int i=3, j=7;
float f1=2.3, f2=4.5;

pf(" Before : i = %d  j = %d \n", i, j);

SWAP(i, j, int);
pf("After ---");

pf(" Befor : f1 f2 ");
SWAP (f1, f2, float);
pf ("After --- ");

}
```

⇒

① `#define swap(a, b) int t; t=a; a=b; b=t`

How to write multiple line.

②
```
#define swap (a b)  int t ; \
                    t = a ; \
→swap (i, j);       a = b ; \
→swap (m, n);       b = t
```
↝ error multiple declaration of int t

③ Ho avoid ||

```
#define swap (a, b) { int t ; \
   →swap (i, j);                }
```
↑ int

→ swap (i, j); ✗ (wrong ans.)
↑ float

④ `#define swap (a, b, type) { type t ; \ }`

→ swap (i, j, int); ✓

→ swap (i, j, float); ✓

#a ⟶ "i"	as a token we can supply anything
#b ⟶ "j"	

PAGE EDGE

Date

* WAP to design a macro to find biggest of two integer.

in " " (string constant), preprocessor cannot perform any task (replacement)

```
#include <stdio.h>
#define BIG(a, b)  if(a > b) . \
                   pf("a is bigger \n" ; \
                   pf(#a " is bigger \n" ; \
                   else
                   pf("b is bigger \n" ; \
                   pf(#b " is bigger \n" ; \

void main()
{
  int i=5, j=10 ;
  BIG (i, j);
}
```

"j" "is bigger \n"

* TOKEN Pasting Operation

symbol : ##

it will take two tokens, concatenate both and make one single token

EX ①
```
#include <stdio.h>
#define TP(a, b)  a##b
void main()
{
  int ij = 20 , k ;
```

:- it will make replacement in the form of string.

```
r = TP(i,j);
r = a##b  ⟶  r = i##j  ⟶  r = ij;

pf("r = %d \n", r);
}
```

NOTE :-

→ pasting "i" and "1.5" does not give a valid preprocessing token
eg. : TP(i, 1.5)

→ we cannot supply one char and one num it will give error
eg. : TP(i, 1)

* Conditional Compilation

using this method we can inform to the preprocessor which part of the program to be placed into .i and which not to be placed so that size of executable file can be reduced.

* WAP which contains multiple main() and inform to preprocesso place any one main() in .i file

```
#include<stdio.h>
#define op 2

#if (op==1)
  void main()
  {
     printf("in main 1... \n");
  }
#elif (op==2)
  void main()
  {
    printf("in main 2... \n");
  }
#else
   void main()
   {
     printf("in main 3... \n");
   }
#endif
```

(?) can we define macro without its body.
yes, we can.

→
```
#include<stdio.h>
#define op
#undef op   → to undefined macro
```

`#ifdef op`
`void main()`
`{ pf("in main 1...");}`
`#else`
`void main()`
`{ pf("in main 2...");}`
`#endif.`
	`#ifndef op` ↖ If op is not defined then condition true
	`#else`
	`#endif`

o/p ↳ in main 2...	o/p ↳ in main 1...

→ Stringizing | ## → token Pasting

* if we define our own header file How to design in such a way that if programer try to include multiple time it will accept one time only.

option ①

at the begning of header file contant we can write #pragma once

EX:

header.h

```
#pragma once
typedef int INTEGER;
int sum (INTEGER, INTEGER);
```

option ②

we can take help of condition compilation

EX:

```
#ifndef op
#define op
 typedef int INTEGER;
 int sum (INTEGER, INTEGER);
#endif
```

* Predefine macro :-

EX:

```
#include <stdio.h>
void main()
{
 pf("%s", __FILE__); → current file name
 pf("%s", __DATE__); → current compilation date
 pf("%s", __TIME__); →  "  time
 pf("%d", __LINE__); → current line no.
}
```

in variable arg. list fincion, va_start(), and va_arg() is macro with arg.

PAGE EDGE

Date :

* File Handling

A named collection of data, stored in secondary storage is called as file.

File is naccessary because.

when a program is terminated, the entire data is lost. Storing in a file will preserved your data even if the program terminated.

if you have to enter a large no. of data, it will take a lot of time to enter them all.

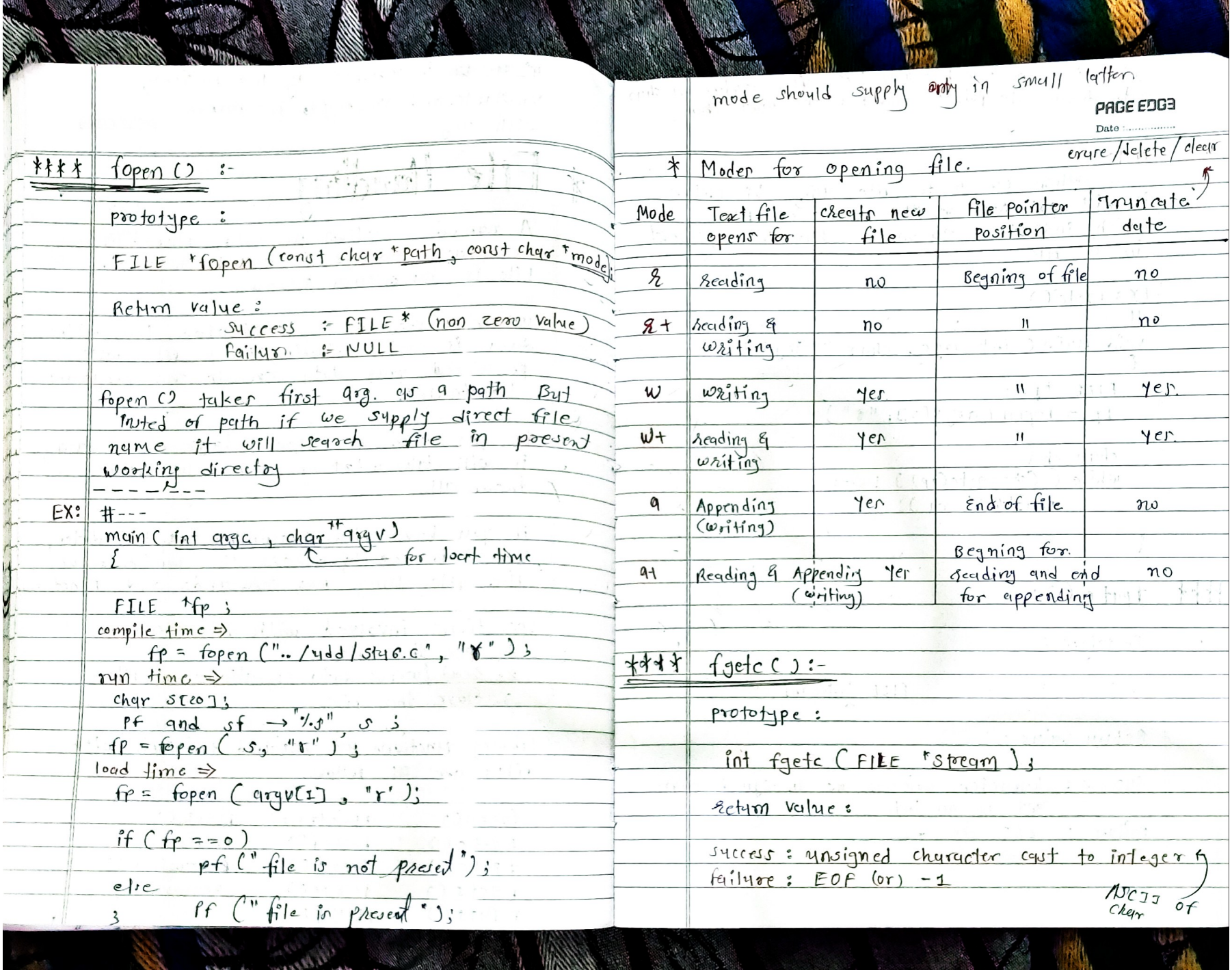
if you have file containing all the data, you can easily access the contents of the file using few commands and function

operations involved with filer are

1. open the file
2. read / write data operations on filen
3. close the file.

Library functions are required to work with files are as follow.

fopen()	fprintf()	fseek()
fgetc()	fscanf()	fclose()
fputc()	fread()	rewind()
fgets()	fwrite()	remove()
fputs()	ftell()	



mode should supply only in small letter

PAGE EDGE

Date

*** fopen() :-

prototype :

FILE *fopen (const char *path, const char *mode);

Return value :

Success : FILE* (non zero value)

Failure : NULL

fopen() takes first arg. as a path But instead of path if we supply direct file name it will search file in present working directory

EX:

```
#---
main( int argc , char **argv )
{                          ↑ for 2nd time

 FILE *fp ;
compile time ⇒
    fp = fopen ("../add/sty6.c", "r" );
run time ⇒
 char s[20];
  pf and sf → "%s" , s ;
 fp = fopen ( s, "r" ) ;
load time ⇒
 fp = fopen ( argv[1] , "r" );

 if ( fp == 0 )
        pf ("file is not present");
 else
 {      pf ("file is present" );
```

* Modes for opening file.

erase / delete / clear

Mode	Text file opens for	creates new file	file pointer position	Truncate data
r	Reading	no	Begning of file	no
r+	Reading & writing	no	"	no
w	writing	yes	"	yes
w+	Reading & writing	yes	"	yes
a	Appending (writing)	yes	End of file	no
a+	Reading & Appending (writing)	yes	Begning for reading and end for appending	no

**** fgetc() :-

prototype :

int fgetc (FILE *stream);

Return value :

success : unsigned character cast to integer

failure : EOF (or) -1

A[3] of char

fgetc() is design in such a way that after reading on char automatically file pointer moves to the next char.

Note :-
when we write file using Vi editor it will put \n at the end of file which indicates no more characters present in a file.

Ex : fgetc()

```
#---
void main(int argc, char **argv)
{
  FILE *fp;
  fp = fopen(argv[1], "r");

  char ch;
  while ((ch=fgetc(fp)) != -1)
        pf("%c", ch);

}
```

**** fputc();

Prototype :

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

Return value :

success : character written as unsigned char cast to an int.
failure : EOF (or) -1

```
#define EOF (-1)
```

PAGE EDGE
Date

* WAP to design userdefine cp command. (or)
* WAP to copy one file into another file (or)
* WAP to copy src file into dest. file.

```
#---
void main(int argc, char **argv)
{
  if(argc != 3)
  {  pf("ગો ધ્યાને \n");
     return;
  }

  FILE *sf, *df;

  sf = fopen(argv[1], "r");
  if(sf==0)
  {
    pf("src file is not present");
    return;
  }

  df = fopen(argv[2], "w");

  char ch;

  while((ch=fgetc(sf)) != EOF)
      fputc(ch, df);

}
```

* WAP to copy src file into multiple destination files.

rewind() → void rewind (FILE *stream);
⇒ rewind() is used to move file pointer at begning of file

⇒ #---
```
void main(int argc, char **argv)
{
 if (argc<3)
 {  pf("ओ आवे");
    return;
 }
 FILE *sf, *df;
 of = fopen (argv[1], "r");
 if(sf==0)
 {  pf(" file is not present ");
    return;
 }
 char ch;
 int i;
 for(i=2; i<argc; i++)
 {
    df = fopen (argv[i], "w");
    while ((ch = fgetc(sf)) != EOF)
          fputc (ch, df);
    rewind (sf);
 }
}
```

* WAP to replace one specific char with any other char in a file
```
#include<stdio.h>
#include<stdlib.h>
void main (int argc, char **argv)
{
  if (argc != 4)
  {  pf("ओ आवे");
     return;
  }
  FILE *fp;
  fp = fopen (argv[1], "r+");
  if (fp==0)
  {
     pf(" file ही नही"); return;
  }
  char ch;
→// find size of file //
  int c=0;
  while ((ch=fgetc(fp)) != EOF)
     c++;
  rewind(fp);
→ allocate dynamic m/s for char array
  char *s = malloc (c+1);
→ copy file content into char array
  int i=0
  while ((ch=fgetc(fp)) != EOF)
  s[i++] = ch;
  s[i] = '\0'; → converted array into string
  rewind (fp);
→ replace char in a string
  for (i=0; s[i]; i++)
  {
    if (s[i] == argv[2][0])
       s[i] = argv[3][0];
  }
→ write updated string in file
  for (i=0; s[i]; i++)
     fputc (s[i], fp);
          → write char by char string into file
  ⇒ → fputs (s, fp);
→ it ~~replace~~ write one string into file
```

fputs() :-

prototype :-

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

Return value :-

success : Return a nonnegative number.
failure : EOF (or) -1

fgets()

prototype :-

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

Return value :-

sucess : fgets() return string on success.
failure : NULL

→ fgetc() is used to read string from the file this string can be entire line of file or some specific portion of a line from file it totally depends on second arg. size

```
#---
void main ()
{
  FILE *fp = fopen ("data", "r");
  char s[100];
```

fgets() ⇒ fgets (s, 100, fp); ⇒ file માંથી 99 char વાંચે. string માં લેવી. 1 byte '\0' માટે. PAGE EDGE Date

જો આ 99 char માં \n આવવાની નથી તેના પહેલાનાં char વાંચે છે.

```
fgets (s, 8, fp);
pf ("%s \n", s); ⇒ abcd123
fgets (s, 100, fp);
pf ("%s \n", s); ⇒ mnop
fgets (s, 2, fp);
pf ("%s \n", s);
```

data

abcd123
mnop

* How to read line by line data from file.

```
#---
void main()
{
  FILE *fp = fopen ("data", "r");
  char s[100];
  while ( fgets (s, 100, fp) )
    pf ("%s", s);
}
```

grep Command

grep command is used to search one specific word in a file, in whichever the line it word is found entire content of that line it will display on a screen.

Syntax :

```
grep word/string filename
```

* WAP to implement user define grep command.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main(int argc, char **argv)
{
  if(argc != 3)
  {  pf("ओ साली");
     return;
  }
FILE *fp;
fp = fopen(argv[1], "r");
if(fp==0)
{
  pf("file is not present \n");
  return;
}
```

logic to find longest line length.

```
char ch;
int c=0, c1 = 0;   c1 which is holding longest line
                   c is used to calc. each line len.
while((ch=fgetc(fp)) != EOF)
{
   c++;
   if(ch=='\n')
   {
      if(c > c1)
         c1 = c;
      c=0;
   }
}
```

PAGE EDGE

Date

```
rewind(fp);
char *s = malloc(c1+1);
while(fgets(s, c1+1, fp))
{
 if(strstr(s, argv[2]))
  pf("%s", s);
}
}
```

**** fscanf()

prototype:

int fscanf(FILE *stream, const char *format, ...);

Return value:

success: Return the number of input items successfully matched and assigned.
Failure: EOF.

**** fprintf()

prototype:

int fprintf(FILE *stream, const char *format, ...);

Return value:

success: Return the number of characters.
failure: Negative value.

~~char by char → fgetc & fputc~~
~~line by line → fgets & fputs~~
~~word by word → fprintf & fscanf~~

EX: ①

fprintf() :-

```
#---
void main ()
{
FILE *fp =fopen("data", "w");
int i = 123456;
fprintf(fp,"%d", i);
fwrite(&i, 4, 1, fp);
```

fscanf() :-

```
#---
void main()
{
FILE *fp = fopen("data", "r");
int i;
fscanf(fp,"%d", &i);
fread(&i, 4, 1, fp);
```

EX ②

```
#---
void main()
{
FILE *fp = fopen("data", "w");
int a[5] = {1, 2, 3, 4, 5}, i;

for (i=0; i<5; i++)
  fprintf(fp, "%d ", a[i]);
```

space is mandatory ↑
આ સ્પેસ આપી file માંથી data scan
કરવું હોય ત્યારે proper scanning માટે.

```
fwrite(a, 4, 5, fp);
```

```
#---
void main()
{
FILE *fp = fopen("data", "r");
int a[5], i;
for (i=0; i<5; i++)
fscanf(fp, "%d", &a[i]);
fread(a, 4, 5, fp);

for (i=0; i<5; i++)
pf("%d ", a[i]);
pf("\n");
}
```

→ fscanf function is used to read formated i/p from the file (or) it will help us to read word by word data from the file bcoz when we call fscanf, it will keep on read data from the file but whenever file pointer has reached to space (or) new line char (or) end of file. function fill stop reading

→ if file is written using fwrite(), then we need to read using fread().

* How to write data in a file

```
#---
typedef struct st
{
int rollno;
char name[20];
float marks;
}v;

void main()
{
v a = {10, "abcd", 45.5};
FILE *fp = fopen("data", "w");

fprintf(fp, "%d %s %f \n",
a.rollno, a.name, a.marks);

}
```

* How to read data from file (multiple student data)

```
#---
===
}v;
void main()
{
  v a[3];
  int i;
  FILE *fp = fopen("data", "r");
  for(i=0; i<3; i++)
  {  fscanf(fp,
      "%d %s %f ",
      &a[i].rollno, a[i].name
      ,&a[i].marks);
  }
  for (i=0; i<3; i++)
  { pf("%d %s %f \n",
      a[i].rollno, a[i].name,
      a[i].marks);
  }
```

* WAP to print the word having digit in another file?

```
#--
int digit_check(const char*);
void main()
{
  FILE *fp = fopen("data", "r");
  FILE *fp1 = fopen("data1", "w");
  char s[10];
```

```
while (fscanf(fp, "%s", s) != EOF)
{
  if (digit_check(s))
    fprintf(fp1, "%s", s);
}
}
int digit_check (const char *p)
{
    int i;
    for (i=0; p[i]; i++)
    {
      if( p[i]>='0' && p[i]<='9" )
      {
          return 1;
      }
    }
    return 0;
}
```

**** fread () :-

Prototype :-

```
size_t fread (void *ptr, size_t size, size_t nmemb,
                                    FILE *stream );
```

Return value :- (fread & fwrite)

success : Return the number of items read.
Failure : 0

**** fwrite ():-

prototype :-

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb
                                    , FILE *stream );
```

PAGE EDGE

Date

* what is diff. b/w fprintf() and fwrite() ?

→ when we write any intger for ex. 123456 using fprintf() it will convert each and every date into ASCII first. and then write it in a file which is treated as ASCII text file but when we write similar integer using fwrite() it will write as it is integer in a file. which is treated as binary file.

→ when we write int for ex. 123456 using fprintf() file size will be more as compare to when we write using fwrite().

→ when we are going to write large no. of datas for ex. array of 5 int. fwrite() is better choise as compare to fprintf()

* What is FILE ?

→ FILE is a typedef name to the pre-defined structure

```
typedef struct _IO_FILE FILE ;
```

⇒ call :-

```
FILE *fp = fopen ("data", "r" );
```

⇒ defination :-

```
FILE *fopen (const char *path, const char *mode)
{
```

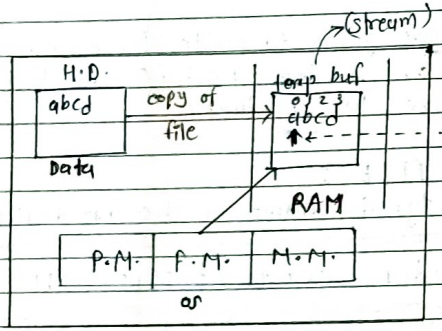
① it will allocate dynamic memory for structure _IO_FILE.

ftell() where current position indicator points at that we can know using ftell()

② it will communicate with O.S. (from O.S. P.M will allocate one temp buffer in RAM (which is nothing but copy of file) is tech. called as stream.

③ it will fill all the structure membs with required info. (in struct one member a variable called current position indicator) which will point at starting of temp buffer.

④ after complition of all the task, on success it will return addr of structure, on failure it will return null.



}

ftell() :

prototype :- long ftell (FILE *stream);

Return value :-

success : Return the current offset (int)
Failure : EOF (or) -1

PAGE EDGE
Date

EX:
```
#include<stdio.h>
void main()
{
FILE *fp = fopen("data","a");
pf("%ld\n", ftell(fp));
}
```
we can use this code to calc. size of file.

fseek() :-

prototype :- int fseek (FILE *stream, long offset, int whence);

Return value :-
success :- 0
failure :- EOF (or) -1

EX:
```
#include<stdio.h>
void main()
{
FILE *fp = fopen("data","r");
fseek(fp, 0, SEEK_END);
pf("%ld\n", ftell(fp));
}
```
we can use this code to calc. size of file.

→ fseek() is used to move file pointer at desired pos in given file, Here we need to take help of third arg. which is nothing but macro there are 3 option of macro ① SEEK_SET which will help us to set file pointer at begning of file. ② SEEK_CUR which will us to set file pointer to current position ③ SEEK_END which will help us to set file pointer at end of a file.

→ from begning of a file (or) End of file (or) current position wherever we want to move file pointer

long offset

SEEK_END ⟶ +ve offset not applicable
SEEK_SET ⟶ -ve offset not applicable.

PAGE EDGE
Date :

second arg. become helpfull which is nothing but long offset, if we want to move forward direction it will be +ve offset, if we want to move in backword direction -ve offset and from that specific position if we dont want to move any where else offset become zero.

fclose() :-

Prototype :- int fclose (FILE *stream);
Return value :-
success :- zero (0)
failure :- EOF (or) -1