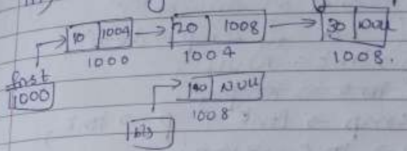
```
if( first = = NULL)
{
 first - temp;
 return;
}

else
 {
  temp → link = first;
  first = tem;
  }
}
```

* Length of linked list.

```
int
~~void~~ length()
{
 int count = 0;
Struct node * temp;
   temp = first;
while (temp != NULL)
{
 count ++;
 [illegible] = temp → link;
 [illegible]; }
```

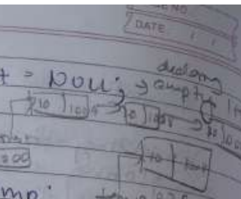
iii) inserting node at given position.



program. (~~as~~ starting same)

```
void atposition()
{
 Struct node * temp, *p;
 int loc, len, i = 1;
printf(" Enter the location");
scanf(" %d " & loc); → location to enter value
 len = length(); → when we use this that code need to be written
if( loc > len)
{
 printf("invaild location");
}
else          → pointer p will point to first element
{ [illegible] p = first;
   while (i < loc)
   {   p = p → link;  } → it will increment
        i ++;
    }
```

```
Struct node *first = NULL; → declaring empty

void append()
{
  Struct node *temp;

  temp = (Struct node *) malloc(sizeof(Struct node));
  printf("enter element to insert");
  Scanf("%d", &temp → data);
  temp → link = NULL;
  if (first == NULL)
  {
   first = temp;
   return;
  }
  else → if it is not empty then insert at end
  {
   Struct node *p;
     p = first;
   while(p → link != NULL)
   {  p = p → link;
   }
```



to create temp

temp is taken to temporary create a node

enter values

the dis inserting at end to that node is NULL

To access and insert values to temp we use temp.data, temp.link

if it is empty then temp point to first

Checking for condition if it is empty

declaring pointer p which holds the address of temp node and we are assigning that p to first and first

traverse node by node till the

```
  p → link = temp;

}
```

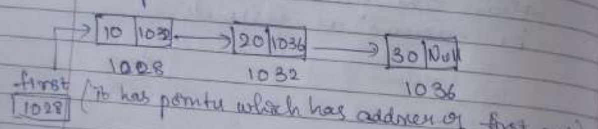
ii) inserting a node at the front end of the list.

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

Struct node *first = NULL;

void insert_front()
{
 struct node * temp;
 temp = (Struct node *) malloc (sizeof(Struct node));
 printf("enter the element to insert");
 Scanf("%d", &temp → data);
 temp → link = NULL;
```

2) Insertion & deletion operations involving array is a ~~ted~~ tedious job, but it is easy in linked list.

1) Singly linked list :- If there exist only one linked field in each node of the list then the linked list is called singly linked lists.

Example :-



(it has pointer which has address of first node)

Use to ~~declare~~ create The node Structure

```
Struct node  → This type of is self referencial structure.
{
  int data;
  Struct node * link;
};                 pointer.
```

uct node *first
= (struct node *) malloc (size of (struct node))
we create dynamic memory allocation with the size

Operation on Singly Linked List.

1) inserting ~~the~~ a node into the list
2) Deleting a node from the list.
3) Display the content of the list.

1) Insertion :-

i) inserting ~~the~~ node at the end of the list

ii) inserting a node at the front end of the list

iii) inserting a node at the given position

i) Inserting a node at the end of the list

Step ⇒ 1 Create a node

2. insert value (data) to fill the link (address) as it is and so Null

3). pointer the address of new node to the previous

example :-

Common for all the list

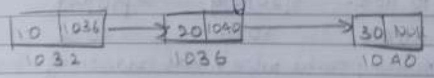
```
Struct node
{
 int data;
 Struct node * link;   };
```

```
printf("delete element is %d", q(*f));
*f = (*f + 1)% max;
*count = -1;
}

void display(int q[], int f, int count)
{
  int i;
  if (count ==0)
  {
    printf("Queue is empty");
    return;
  }
printf("Content of the queue is ");
for (i=1 ; i<=count ; i++)
 {
  printf(" %d", q[f]);
  f = (f+1)% max;
 }
}
```

Unit : 3

Linked list : It is a collection 0 or more nodes where each node is conne-cted to one or more nodes. Each nodes has Two fields. that is data field or (info field). and another is link field.

Example : Assume that i want store 10, 20, 30 element using linked list than the pictorial representation can be viewed as follows.



Types of Linked List.

1) Singly linked list.
2) Doubly linked list
3) Circular singly linked list.

IMP Advantages of linked list over arrays.

1) (Operation insertion, deletion)
2) Size of array is fix where as in linked list it is dynamic

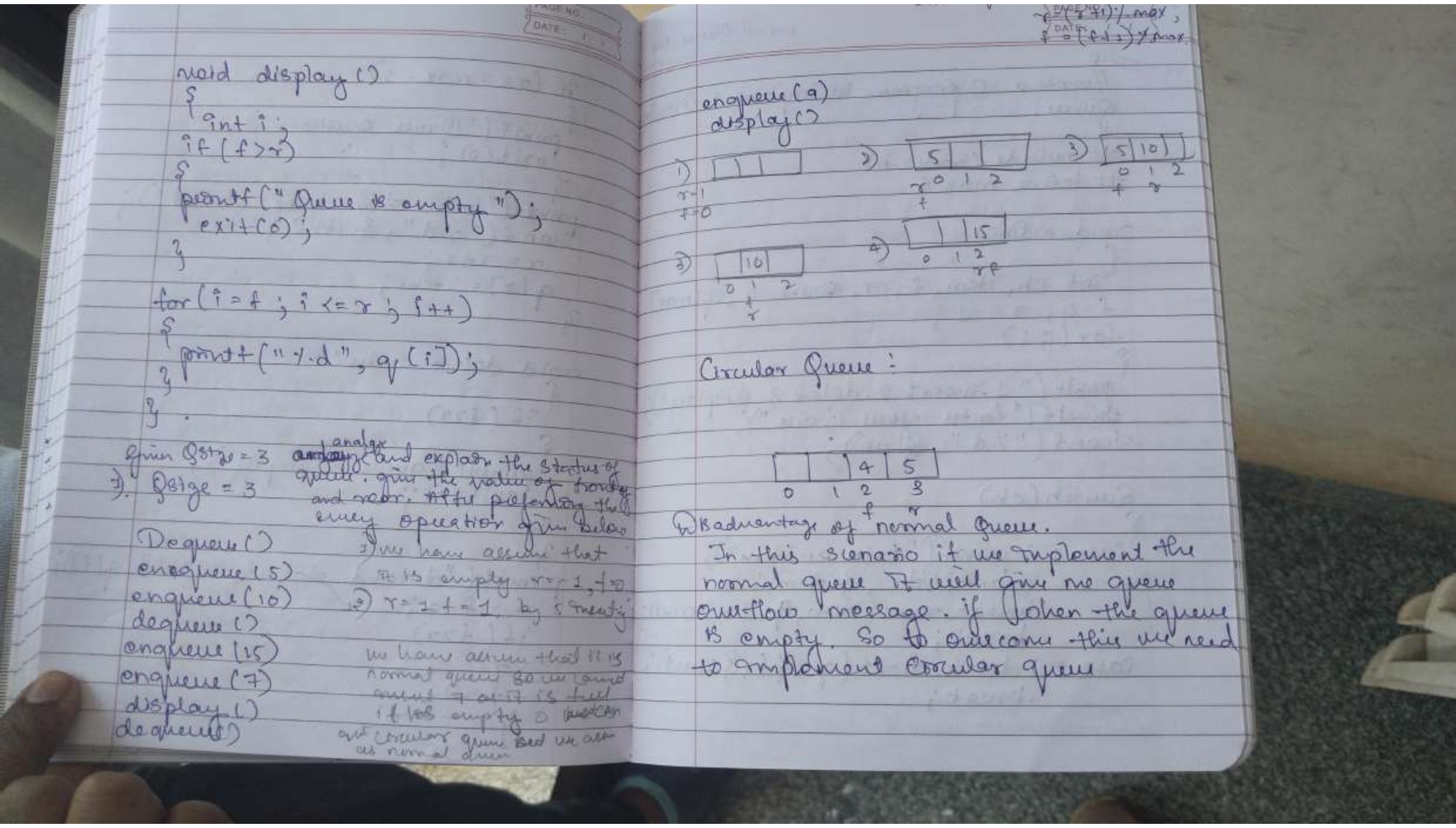
normal Queue for [illegible]

v. Imp

//write a C program to implement circular Queue.

```
#include <stdio.h>
#define max 5

void main()
{
 int ch, item, f, r, count =0 , q[max]
 f=0; r=-1; → empty queue
 for( ; ; )
 {
  printf("1.insert 2.delete 3. display");
  printf("Enter your choice");
  scanf("%d", &ch);

  Switch(ch)
  {
  case 1 : printf("Enter an item");
           scanf("%d", &item);
           insert rear(item, q, &r, &count);
                                 address of rear & count
           break;
  case 2 : delete_front(q, &f, &count);
           break;
```

```
case3 : display (q, f, &count);
        break;
default : exit(0);    count is always no of
                      element present in queue
} }

insert_rear (int item, int q[], int *r,
             int *count)
{         * because we want value and we are passing
                                           address
  if (*count == max)                       & count
  { printf("Queue overflow");
    return;
  }    if its putting 0
  *r = (*r+1) % max;   %→ modulus will give remainder.
                                            5 / 5 = 0
  q[*r] = item;                 It will check the
  *count += 1;                  rear and rear+1
  }                             and that will be
                                % of remainder and
                                will take the index
                                and it will store to
                                that

void delete_front (int q[], int *f,
                   int * count)
{
  if (*count==0)
  {
   printf("Queue underflow");
  } return;
```



```
void display()
{
  int i;
 if (f>r)
 {
 printf("Queue is empty");
   exit(0);
 }

for (i=f ; i<=r ; i++)
 {
  printf("%d", q[i]);
 }
}.
```

Given Qsize = 3 ~~analy~~ analyze and explain the status of
1). Qsize = 3 queue. give the value of front and rear. After performing the every operation given below

Dequeue()
enqueue(5)
enqueue(10)
dequeue()
enqueue(15)
enqueue(7)
display()
dequeue()

1) we have assume that it is empty r=-1, f=0
2) r=1 f=1 by 5 inserting

we have assume that it is normal queue so we cannot insert 7 as it is full if was empty 0 [illegible] as circular queue and we also as normal queue

r = (r+1) % max ,
f = (f+1) % max

enqueue(9)
display()

1)

r=-1
f=0

2)

5		
0	1	2
r f		

3)

5	10	
0	1	2
f	r	

3)

	10	
0	1	2
	f r	

4)

		15
0	1	2
		r f

Circular Queue :

		4	5
0	1	2	3
		f	r

Q) Disadvantage of normal Queue.
In this scenario if we implement the normal queue It will give me queue overflow message. if when the queue is empty. So to overcome this we need to implement circular queue

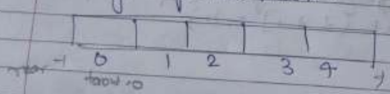
```
f=0; r=-1

for (;;)→ To display menu again & again

{ printf("1. insert \n 2. delete 3. display");
 printf("Enter your choice ");
 scanf("%d", & choice);

 Switch (choice)
 {
  Case 1 : insert_queue();
           break;
  case 2 : delete_queue();
           break;
  case 3 : display();
           break;
  }
 }
}

void insert_queue()
{ int item;
```

```
if (r==max-1)
{
 printf("Queue overflow ");
  exit(0);
}
printf("Enter item to be inserted ");
scanf("%d", &item);
  r=r+1;
  q[r]=item;
}

void delete_queue()
 {
  if (f>r)
  {
   printf("queue is empty");
   exit(0);
  }
 }
printf("deleted element is %d", q[f]);
 f = f+1;
 if(f>r)
   r = -1;
   f=0;
 }
```

fifo

Queue :- Queue is a data structure where ele-ments can be inserted from one end and element are deleted from other end. The end at which the elements are added is called rear end. and end from which the element are deleted is called front end. It is also called as fifo data structure.

Array implementation.



Three operation {
insertion
deletion.
display

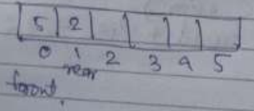
rear = -1   front = 0

To insert in queue first increment rear then insert front will be

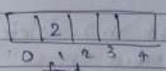
To delete pop first element and increment the front

whenever Queue is empty front will be greater than rear

* Insertion (Enqueue)



* deletion (dequeue)



* Display().

Imp // write a C program to implement the Queue using Array.

f -> front
r -> rear
q -> array

```
#include <stdio.h>
# define max 10

int f, r, q[max];

void main()
{
 int choice;
```

PAGE NO.
DATE: / /

#write a C program to evaluate postfix expression.

```
# include <stdio.h>
# include <math.h>
# define size 30
double eval (char exp [])
{
 double eval (char exp [])
 {
 double op1, op2, stk [Size];
 Char Symb;
  int i, t;
  i = 0;
  t = -1;
  )

do
{
 Symb = exp [i];
 if (Symb > '0' && Symb < '9')
  Stk [t++] = Symb - '0';
                          )
else
{
 op2 = Stk [t--];
 op1 = Stk [t--];
```

array with name stk

top

PAGE NO.
DATE: / /

```
Switch (symb)
{
 Case '+' : Stk [t++] = op1 + op2;
 Case '-' : Stk [t++] = op1 - op2;
 Case '*' : Stk [t++] = op1 * op2;
 Case '/' : Stk [t++] = op1 / op2;
 Case '$' : b = pow (op1, op2)
     Stk [t++] = b;

 }
  }
  i ++;
  )
} while (exp[i] != '\0');
  return (Stk [t]);
                  )
}

void main ()
{
 Char expr [size] ;
 printf ("Enter postfix expression");
Scanf ("%s", expr);
printf (" The result is %.lf ") eval(exp))
```

end of string

top value of Stack, after performing operations

1	2	+	~~6~~
0	1	2	3

write a C program to evaluate postfix expression.

```
# include <stdio.h>
# include <math.h>
#define size 30
double eval (char exp [])
{
 double eval (char exp [])
 {
 double op1, op2, stk [size];
 char symb;
  int i, t;
   i = 0;
   t = -1;

do
{
 symb = exp[i];
 if (symb > '0' && symb < '9')
  stk [t++] = symb = '0';

else
{
 op2 = stk [t--];
 op1 = stk [t--];
```

exp given

arrays with name stk
t = top

array tracking

0 to 9 number

This is used to recognise ASCII value of that char so it is written [illegible]

```
switch (symb)
{
 case '+' : stk [t++] = op1 + op2;
 case '-' : stk [t++] = op1 - op2;
 case '*' : stk [t++] = op1 * op2;
 case '/' : stk [t++] = op1 / op2;
 case '$' : b = pow (op1, op2)
      stk [t++] = b;

 }
  }
 i++;

} while (exp[i] != '\0');
 return (stk [t]);
}

void main ()
 {
  char expr [size];
  printf ("Enter postfix expression");
 scanf ("%s", expr);
 printf ("The result is %.lf") eval(exp));
```

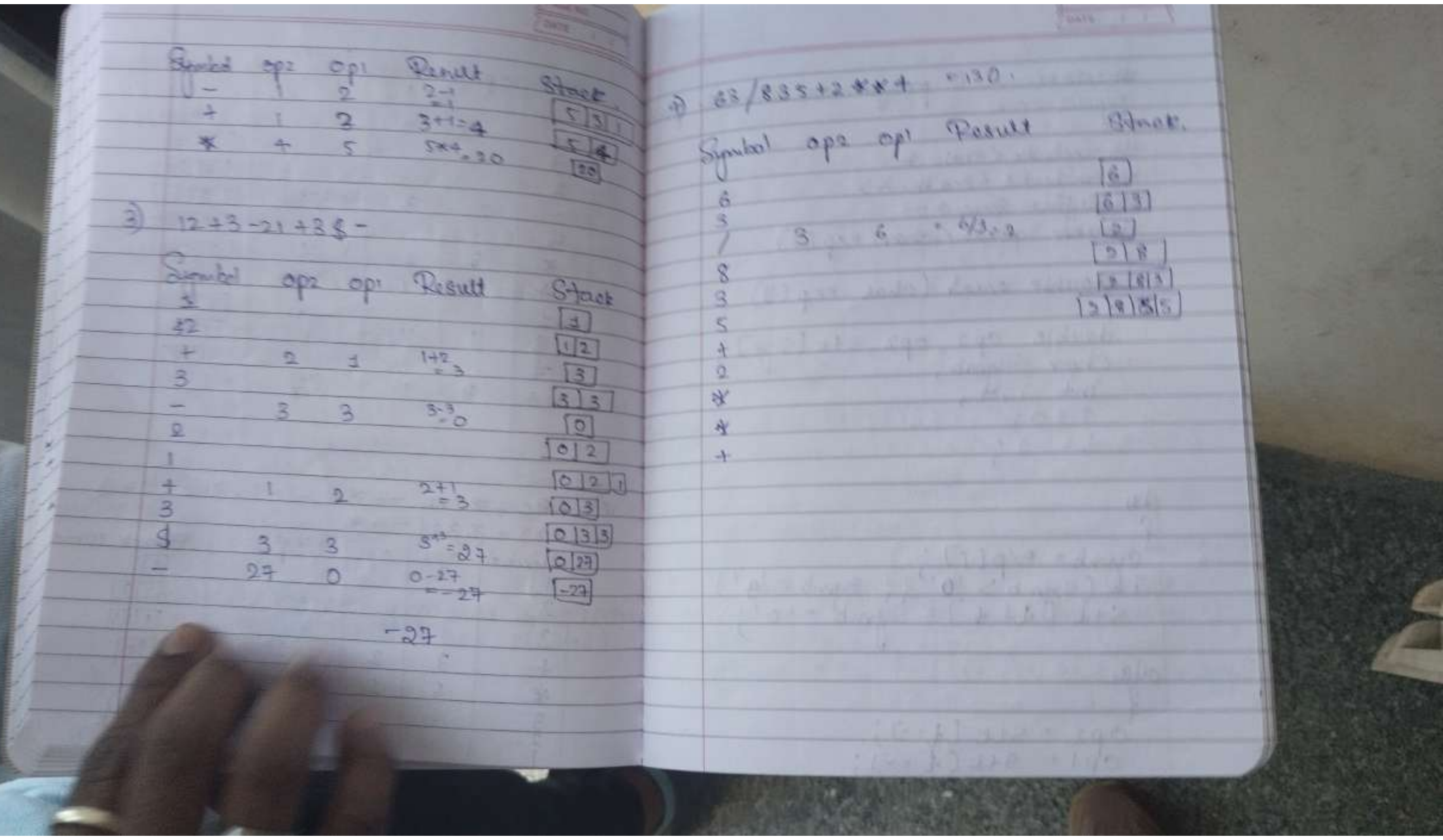
This stack will scan all the character till null value in string

end of string

(string character null which indicate it is end of char

top value of stack after performing operation

1	2	+	~~6~~	symb
0	1	2	3	



Symbol	op2	op1	Result	Stack
-	1	2	2-1 =1	5 | 3 | 1
+	1	3	3+1=4	5 | 4
*	4	5	5*4=20	20

3) 12+3-21+3$-

Symbol	op2	op1	Result	Stack
1				1
2				1 | 2
+	2	1	1+2 =3	3
3				3 | 3
-	3	3	3-3 =0	0
2				0 | 2
1				0 | 2 | 1
+	1	2	2+1 =3	0 | 3
3				0 | 3 | 3
$	3	3	3^3=27	0 | 27
-	27	0	0-27 =-27	-27

-27

4) 63/835+2**+ =130.

Symbol	op2	op1	Result	Stack.
6				6
3				6 | 3
/	3	6	6/3=2	2
8				2 | 8
3				2 | 8 | 3
5				2 | 8 | 3 | 5
+				
2				
*				
*				
+				

2) Evaluation of Postfix expression.

Steps to evaluation Postfix expression

1) Scan the symbol from left to right

2) ~~Sto~~ If scaned symbol is an operand then push it on to the stack.

3) If scanned symbol is an operator the pop two element from the stack first pop element in opreand ~~to~~ 2 and second pop element in operand 1.

op2 = S[top--];

op1 = S[top--];

4) perform the indicated operation as ~~result equal to~~

result = op1 operation op2

5) push the result on to the stack

6) Repeat the above processor till end of ~~the~~ the input is encountered ~~end of the~~

Eg :

evaluate following postfix expression using stack (tabulation method)

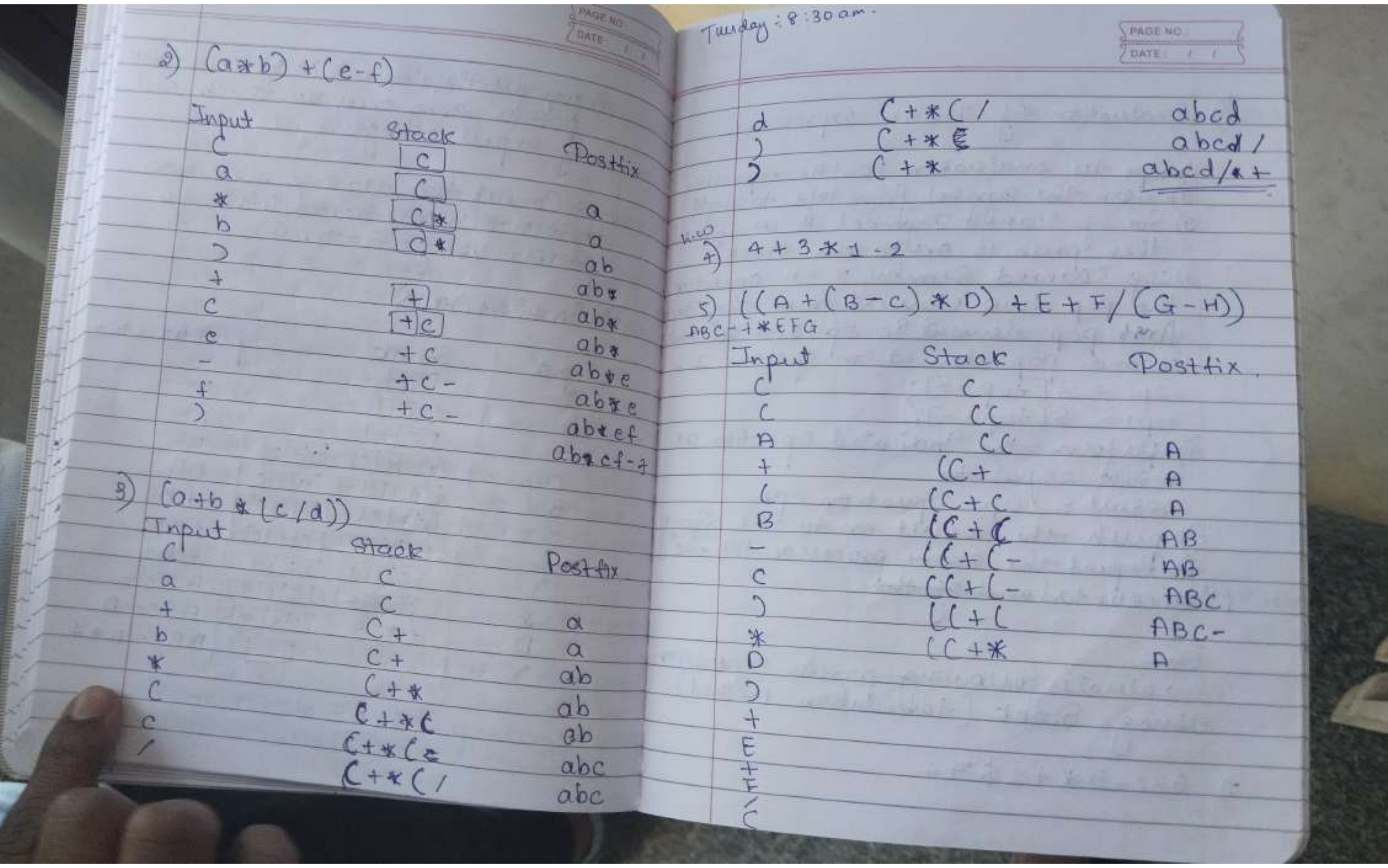
1) 632-5*+1$7+

Symbol	op2	op1	Result = op1 op op2	Stack content
6				$\boxed{6}$
3				$\boxed{6 \,\vert\, 3}$
2				$\boxed{6 \,\vert\, 3 \,\vert\, 2}$
-	2	3	R = 3-2 = 1	$\boxed{6 \,\vert\, 1}$
5				$\boxed{6 \,\vert\, 1 \,\vert\, 5}$
*	5	1	R = 5*1 = 5	$\boxed{6 \,\vert\, 5}$
+	5	6	= 6+5 11	$\boxed{11}$
1				$\boxed{11 \,\vert\, 1}$
$	1	11	= 11^1 = 11	$\boxed{11}$
7				$\boxed{11 \,\vert\, 7}$
+	7	11	= 11+7 18	$\boxed{18}$

2) ABC+*CBA-+* , A = 1 , B = 2 , C = 3

123+*321-+*

Symbol	op2	op1	Result = op1 op op2	Stack content
1				$\boxed{1}$
2				$\boxed{1 \,\vert\, 2}$
3				$\boxed{1 \,\vert\, 2 \,\vert\, 3}$
+	3	2	= 2+3 5	$\boxed{1 \,\vert\, 5}$
*	5	1	= 5*1 = 5	$\boxed{5}$
3				$\boxed{5 \,\vert\, 3}$
2				$\boxed{5 \,\vert\, 3 \,\vert\, 2}$
1				



2) (a*b) + (e-f)

Input	Stack	Postfix
(	(	
a	(	a
*	(*	a
b	(*	ab
)		ab*
+	+	ab*
(	+(	ab*
e	+(	ab*e
-	+(-	ab*e
f	+(-	ab*ef
)		∴ ab*ef-+

3) (a+b*(c/d))

Input	Stack	Postfix
(	(	
a	(	a
+	(+	a
b	(+	ab
*	(+*	ab
(	(+*(	ab
c	(+*(c	abc
/	(+*(/	abc

Tuesday : 8:30 am.

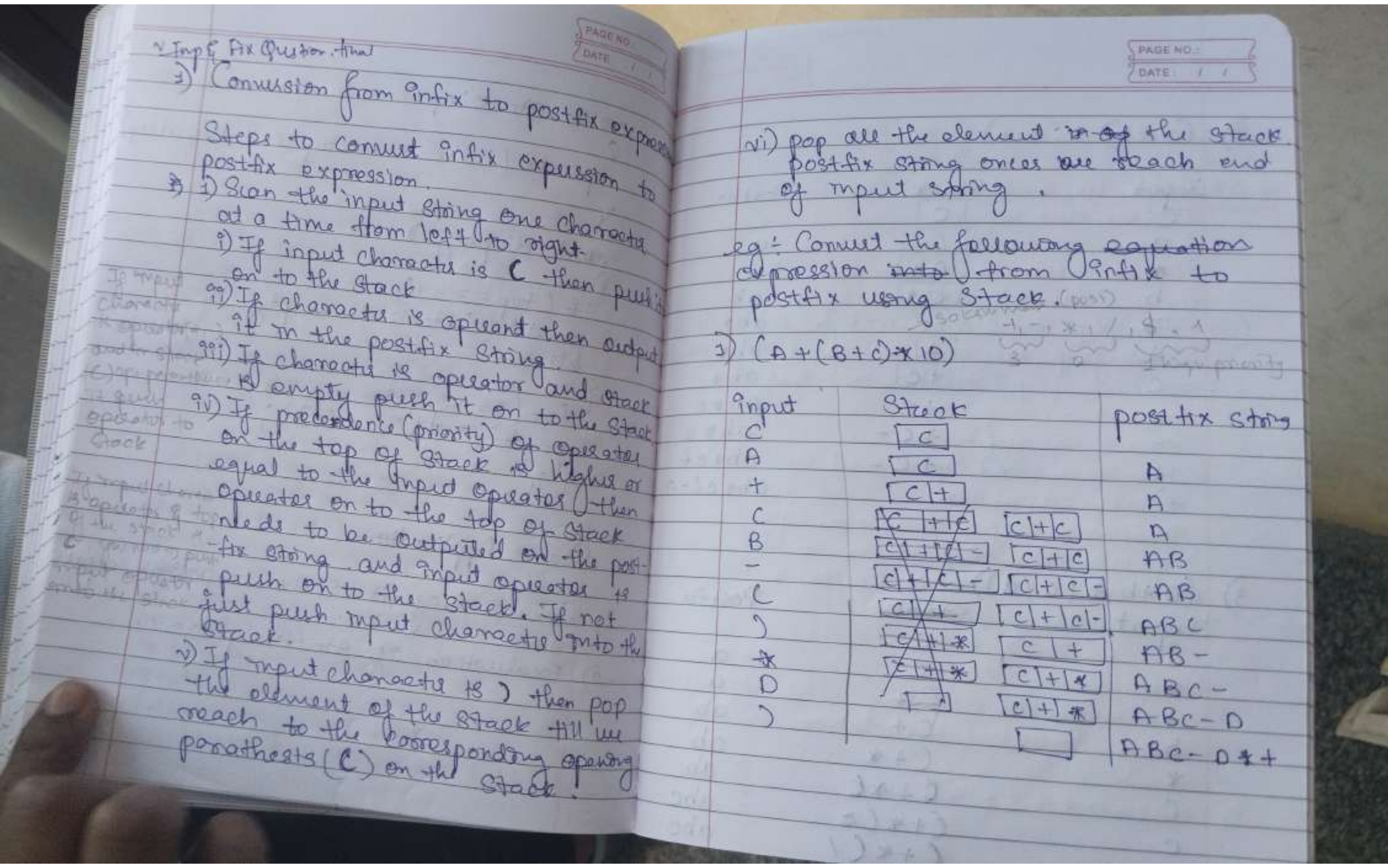
d	(+*(/	abcd
)	(+*(	abcd/
)	(+*	abcd/*+

h.w

4) 4+3*1-2

5) ((A+(B-C)*D)+E+F/(G-H))
ABC-+*EFG

Input	Stack	Postfix.
(	(	
(	((	
A	((	A
+	((+	A
(	((+(	A
B	((+(	AB
-	((+(-	AB
C	((+(-	ABC
)	((+(	ABC-
*	((+*	A
D		
)		
+		
E		
+		
F		
/		
(		



~ Inp & fix Question final

PAGE NO.
DATE / /

1) Conversion from infix to postfix expression

Steps to convert infix expression to postfix expression.

1) Scan the input string one character at a time from left to right.

i) If input character is (then push it on to the stack

ii) If character is operand then output it in the postfix string.

iii) If character is operator and stack is empty push it on to the stack.

iv) If precedence (priority) of operator on the top of stack is higher or equal to the input operator then operator on to the top of stack needs to be outputted on the postfix string and input operator is push on to the stack. If not just push input character into the stack.

v) If input character is) then pop the element of the stack till we reach to the corresponding opening parathesis (() on the stack.

PAGE NO.
DATE / /

vi) pop all the element ~~in~~ ~~of~~ the stack postfix string onces we reach end of input string.

eg :- Convert the following ~~equation~~ expression ~~into~~ from infix to postfix using stack.

1) (A+(B+C)*10)

input	Stack		postfix string
(	(		
A	(		A
+	(+		A
(	~~(+ (~~	(+ (	A
B	~~(+ (-~~	(+ (	AB
-	~~(+ (-~~	(+ (-	AB
C	~~(+~~	(+ (-	ABC
)	~~(+ *~~	(+	AB -
*	~~(+ *~~	(+ *	ABC -
D	~~[ ]~~	(+ *	ABC - D
)			ABC - D * +

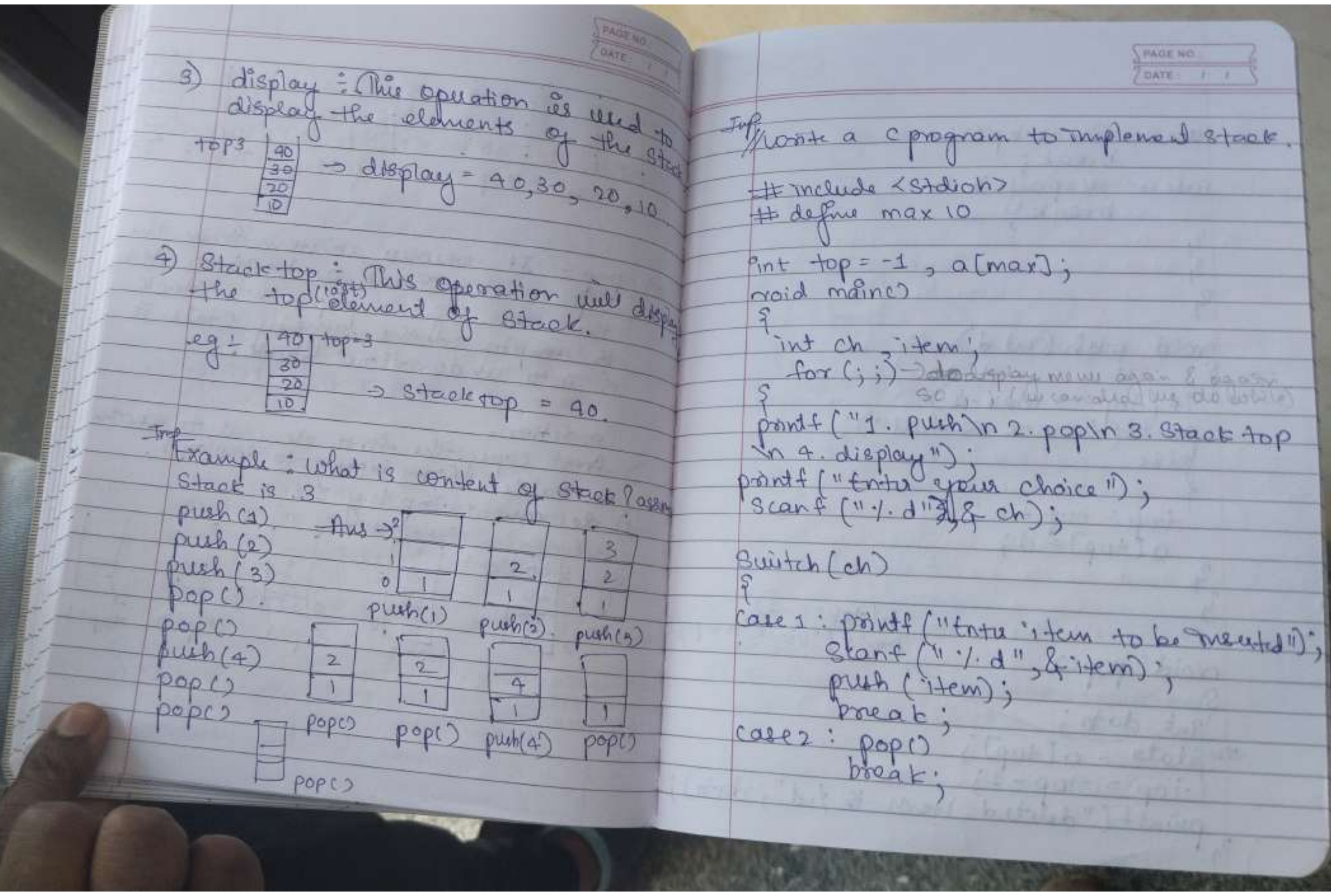
```
case 3 : Stacktop();
         break;
case 4 : display()
         break;
  }
  }
  }

void push (int d)
{
 if (top = = max - 1)
 printf(" stack is full ");
 else
 {
   top = top + 1 ;
   a[top] = d;
 }
}

void pop()                    if (top = = -1)
{                             printf ("Stack is empty")
 int data;
 data = a[top];
 top = top - 1;
printf ("deleted item is %d", data);
```

```
void stacktop ()
{
 printf (" The Stack top is % d", a[top])
}
void display()
{
 if (top = = -1)
 printf ( " Stack is empty ");
 else
 {
  for (i = top ; i> =0 ; i --)
  printf (" % d" ; a (i]);
 } }
```

Stack Applicetion

1) Conversion ~~of~~ from infix to postfix experession

2) Evaluation of ~~stack~~ expussions.



3) display : This operation is used to display the elements of the stack.

top3

40
30
20
10

→ display = 40, 30, 20, 10.

4) Stack top : This operation will display the top (last) element of stack.

eg :

40	top=3
30	
20	
10	

→ stacktop = 40.

Imp
Example : What is content of stack? assum Stack is 3

push(1)
push(2)
push(3)
pop()
pop()
push(4)
pop()
pop()

Ans →

	push(1)	push(2)	push(3)	pop()	pop()	push(4)	pop()	pop()
2			3					
1		2	2	2	2	4		
0	1	1	1	1	1	1	1	

Imp
// write a c program to implement stack.

```
# include <stdio.h>
# define max 10

int top = -1 , a[max];
void main()
{
 int ch , item;
  for (;;) -) ~~do~~ display menu again & again so ;; (we can also use do while)
{
 printf("1. push\n 2. pop\n 3. Stack top
 \n 4. display");
printf("Enter your choice");
 scanf("%d", & ch);

Switch(ch)
{
case 1 : printf("Enter item to be inserted");
         Scanf("%d", &item);
         push (item);
         break;
case 2 : pop()
         break;
```

Unit ÷ 02.

Stack ÷ Stack is a data structure which is also called as LIFO Last In First Out in which all the insertions and deletions are ~~[illegible]~~ restricted to 1 end called Top.

Here, the element which is ~~first~~ entred last into the stack is the first to come out of the stack.

Basic stack operations ⇒ push, pop, display, Stacktop.

1) push ÷ This operation add the items into the Stack.

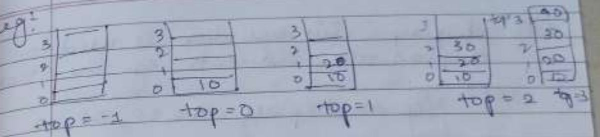
When Top is -1 that indicates the Stack is empty.

If you want to ~~instead~~ insert element to stack when Stack is empty that time increment top by 1. before it -1 then it will be 0. then 1 2 3 all...

first check whether Stack is full (condition) Condition.

1) Check Stack overflow } by checking this two condition then only we can insert values.
2) Top increment.

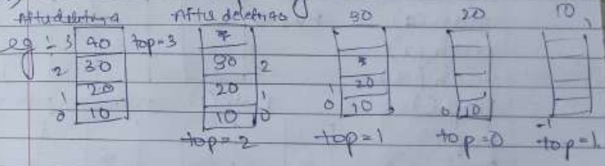
eg:

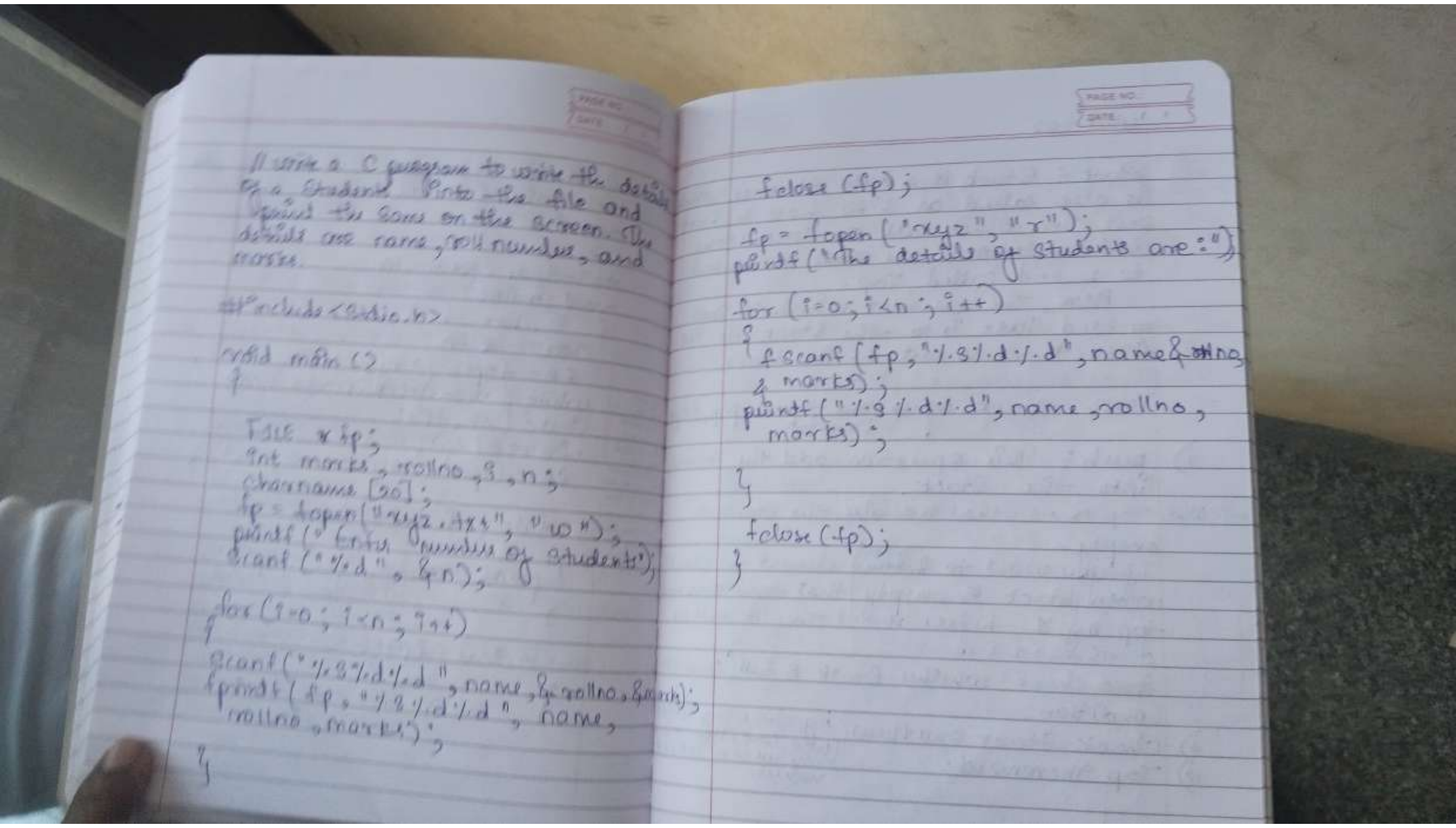


2) pop ÷ It removes an item from the top of the stack and return its to the user. If we pop when Stack is ~~empty~~ in the empty state it is into under flow (empty) state.

Condition.

1) first copy the first element to another variable. (copy because to display).
2) decrement top by 1.

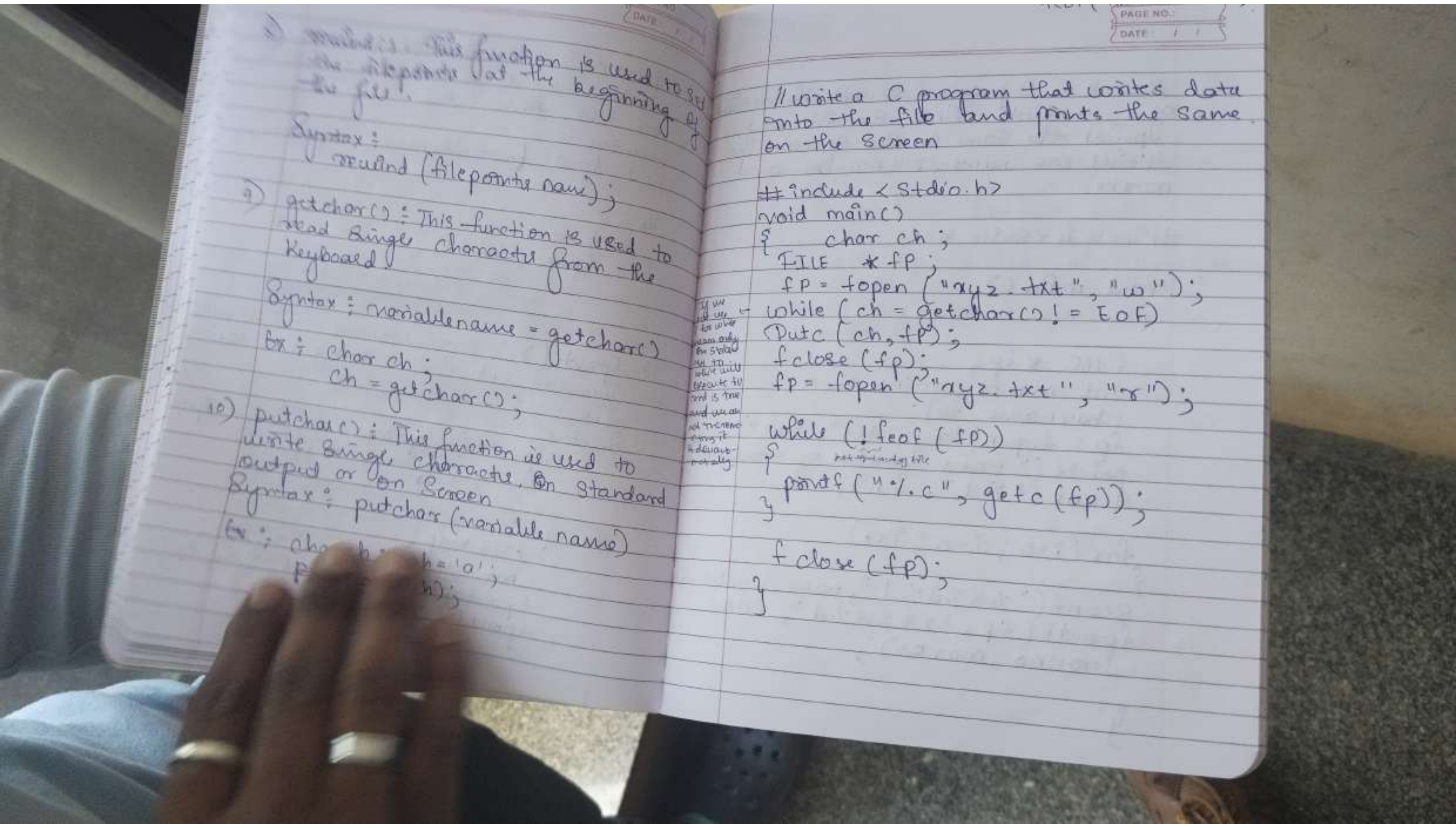




// write a C program to write the details of a student into the file and print the same on the screen. The details are name, roll number, and marks.

```
#include<stdio.h>

void main ()
{

 FILE *fp;
 int marks, rollno, i, n;
 charname [50];
 fp = fopen("xyz.txt", "w");
 printf(" Enter number of students");
 scanf (" %d", &n);

for (i=0; i<n; i++)
{
 scanf(" %s%d%d", name, &rollno, &marks);
 fprintf(fp, "%s%d%d", name,
  rollno, marks);
}
```

```
 fclose (fp);

fp = fopen("xyz", "r");
printf("The details of students are:");

for (i=0; i<n; i++)
{
 fscanf(fp, "%s%d%d", name&~~roll~~no,
 & marks);
printf("%s%d%d", name, rollno,
 marks);

}

 fclose(fp);
}
```



8) rewind(): This function is used to set the filepointer at the beginning of the file.

Syntax:
  rewind (filepointer name);

9) getchar() : This function is used to read single character from the keyboard.

Syntax : variablename = getchar()

Ex : char ch ;
    ch = getchar();

10) putchar() : This function is used to write single character on standard output or on screen

Syntax : putchar (variable name)

Ex : ch[illegible] ch='a';
  p[illegible]h);

PAGE NO.
DATE / /

// write a C program that writes data into the file and prints the same on the screen

```
#include <stdio.h>
void main()
{    char ch ;
 FILE  *fp ;
 fp = fopen ("xyz.txt", "w");
 while (ch = getchar() != EOF)
 Putc (ch, fp);
 fclose (fp);
 fp = fopen ("xyz.txt", "r");

 while (!feof (fp))
 {
  printf ("%c", getc(fp));
 }

 fclose (fp);
}
```

If we did use [illegible] for while [illegible] only the statement [illegible] to while will execute till cond is true and we are not incrementing it [illegible]

PAGE NO.
DATE / /

string

4) fgets() :- This function is used to read String from the file.

Syntax :-

```
fgets (String, length, filepointer);
```

Example :-
```
char Str[10];
fgets (Str, 10, ptr);
```

note :- read from the file where the pointer is pointing and reads 10 characters & Stores in Str.

i.e character

5) fgetc() :- This function is used to read Single character from the Specified Stream

Syntax :-

```
fgetc(file pointer);
```

example :-

```
fgetc (fp);
```

PAGE NO.
DATE / /

6) fputc() :- This function is used to write a single character at a time to the given file.

Syntax :-

```
fputc (variable name, filepointer);
```

Ex :-

```
char ch;
fputc (ch, fp);
```

character that need to insert in given file

7) fprintf() :- This function is used to write character set that is strings into the file.

Syntax :-

```
fprintf (filepointername, "format Speci-
-fier", argument list);
```

Example :-

```
FILE * fp;
char Str [10] ; Str = "xyz");
fp = fopen ("xyz.txt", "w");
fprintf (fp, "%s", Str);
```

JMS

* file handling in c.

Steps in file handling.
1) declare a file pointer.
2) open a file in specified mode.
3) perform the required operation.
4) Close the file.

* file handling function.
1) fopen() :
~~Sy~~
This function is used to creat a open file or open existing file in c.
Syntax :
filepointername = fopen ("filename", "mode");

mode { w → write
r → read
a → append.

Example : FILE * fp;
fp = fopen ("xyz.txt", "r");

(This should done first before any functn)

* Declaring a file pointer :
Syntax :
FILE * pointername;

DATE: / /

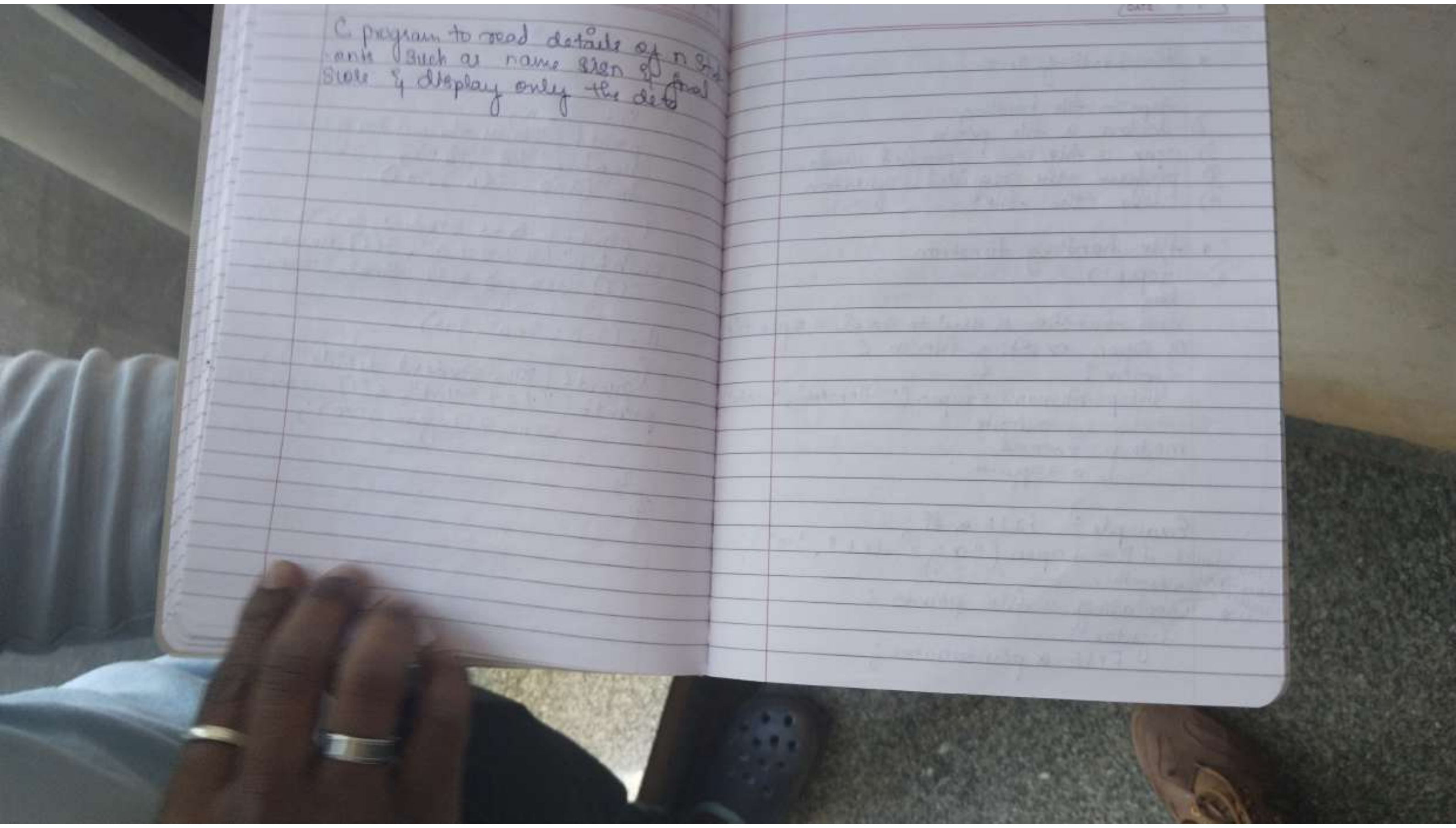
2) fclose() : This function is used to close the file.
Syntax : fclose (filepointername);

Ex :
FILE * fp;
fclose (fp);

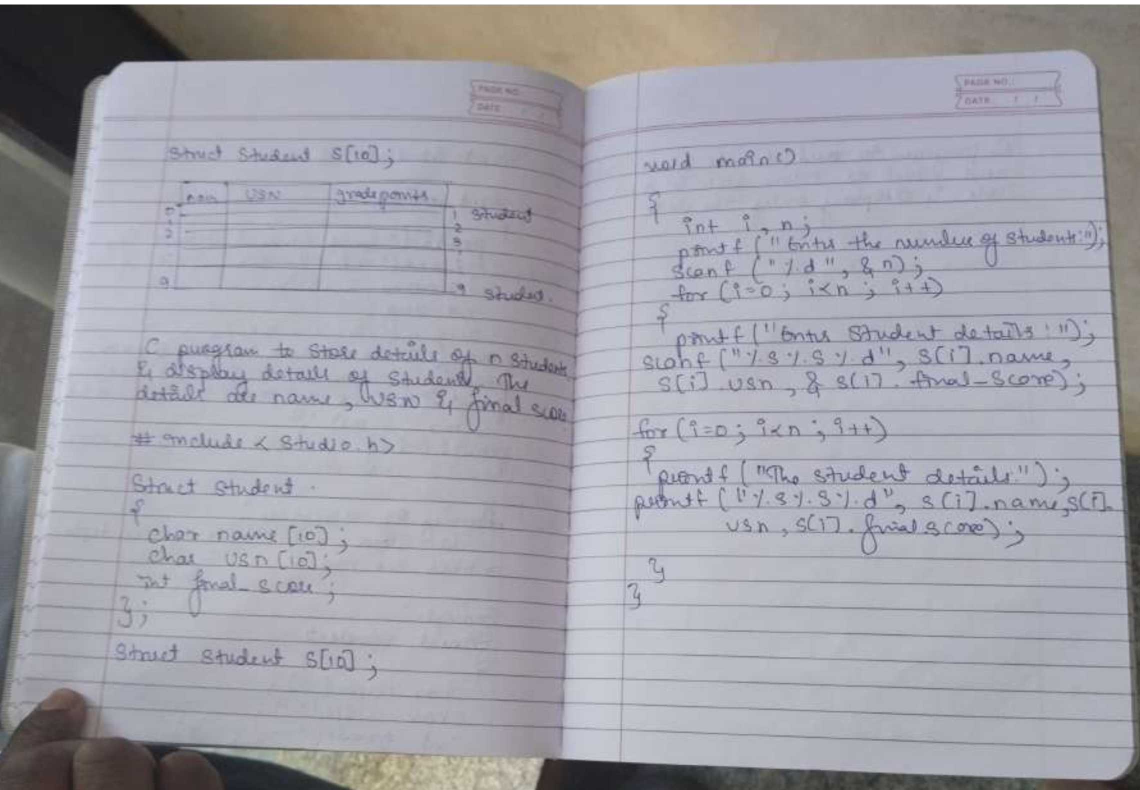
3) fscanf () : This function is used read character set that is strings from the file. It returns EOF (end of file) when all the contains of the file are read by it.

Syntax :
fscanf (filepointername, "format specifiers", argument list);

example :
FILE * fp;
int c;
fp = fopen ("data.txt", "r");
fscanf (fp, "%d", c);



C program to read details of n Students Such as name usn & final Score & display only the det



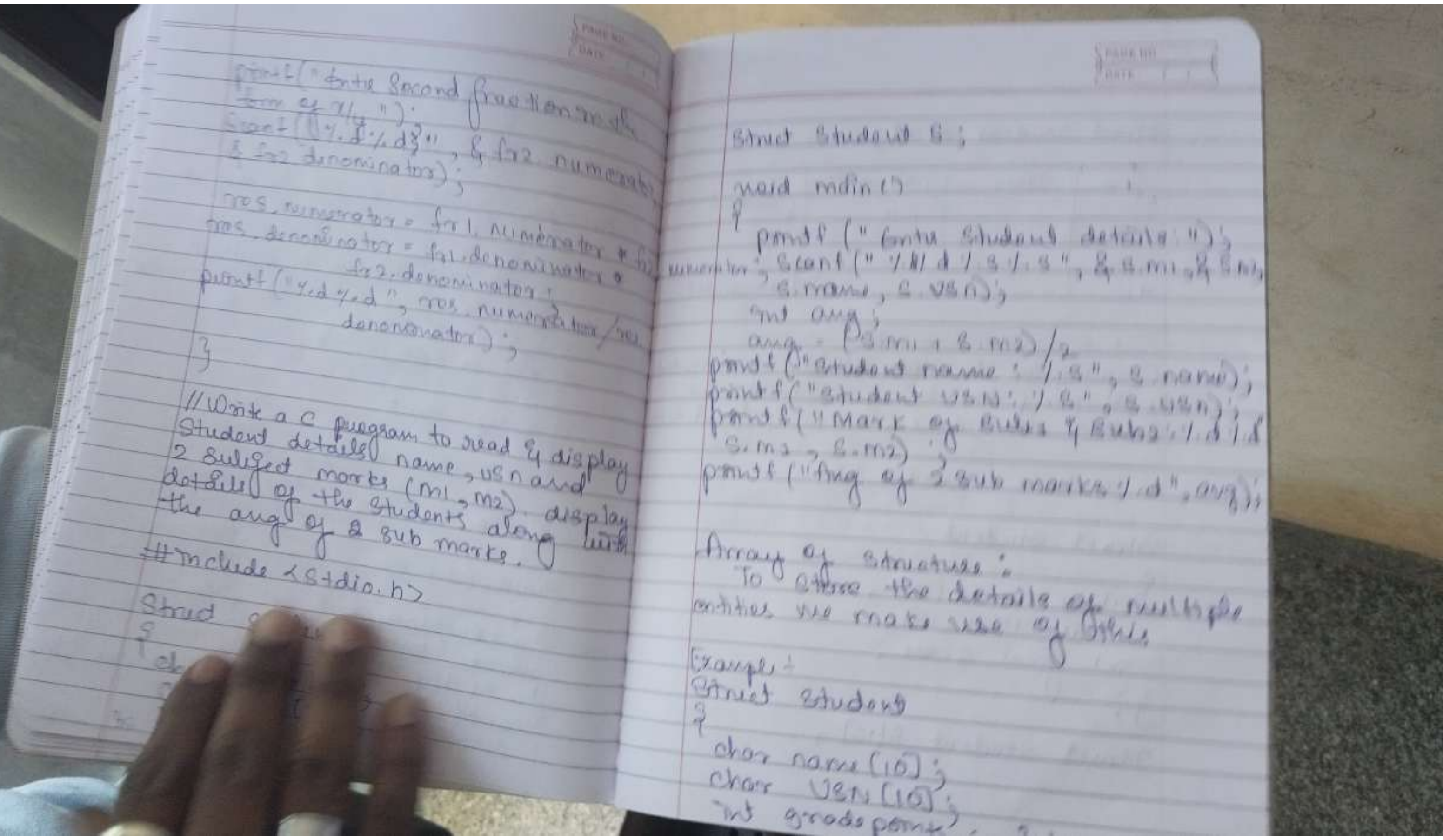
Struct Student S[10];

	name	USN	gradepoints	
0	-			1 Student
1	-			2
2	-			3
9				9 student.

C pugram to Store details of n Students & display details of Students. The details are name, USN & final score

```
# include < Studio.h>

Struct Student .
{
 char name [10];
 char usn [10];
int final_score;
};

Struct student S[10];
```

```
void main()
{
  int i, n;
  printf ("Enter the number of students:");
  Scanf ("%d", &n);
  for (i=0; i<n; i++)
 {
  printf ("Enter Student details : ");
 scanf ("%s %s %d", S[i].name,
  S[i].usn, & s[i]. final_score);

for (i=0; i<n; i++)
{
 printf ("The student details");
printf ("%s%s%d", s[i].name, s[i].
        usn, s[i]. final score);

 }
}
```



```
printf("Entre Second fraction in the form of x/y ");
Scanf("%d%d", & fr2.numerator,
& fr2.denominator);

res.numerator = fr1.numerator * fr2.numerator;
res.denominator = fr1.denominator *
            fr2.denominator;
printf("%d %d", res.numerator/res.
            denominator);
}
```

//Write a C program to read & display Student details(name, usn and 2 Subject marks (m1, m2). display details of the students along with the avg of 2 sub marks.

```
#include <stdio.h>
Struct
{
ch
```

```
Struct Student S;

void main()
{
  printf("Entre Student details ");
  Scanf("%d%d%s%s", &s.m1, &s.m2,
   s.name, s.USN);
  int avg;
  avg = (s.m1 + s.m2)/2
printf("Student name: %s", s.name);
printf("Student USN: %s", s.USN);
printf("Mark of Sub1 & Sub2: %d %d"
 s.m1, s.m2);
printf("Avg of 2 sub marks %d", avg);
```

Array of Structures:
To store the details of multiple entities we make use of this.

Example:

```
Struct Student
{
 char name[10];
 char USN[10];
 int gradepoint;
```

2) Type Declaration using typedef

Syntax :-

```
        keyword
typedef    Struct  -> keyword.
  {
     field list ;
  } TYPE ; (It should be written in capital
    variable name        word always).
```

example :-

```
typedef struct
{
  char name [10] ;
  char id [10] ;
  int gradepoints ;
} STUDENT ;
```

* Structure variable declaration

STUDENT S ; (before we were writing struct students)

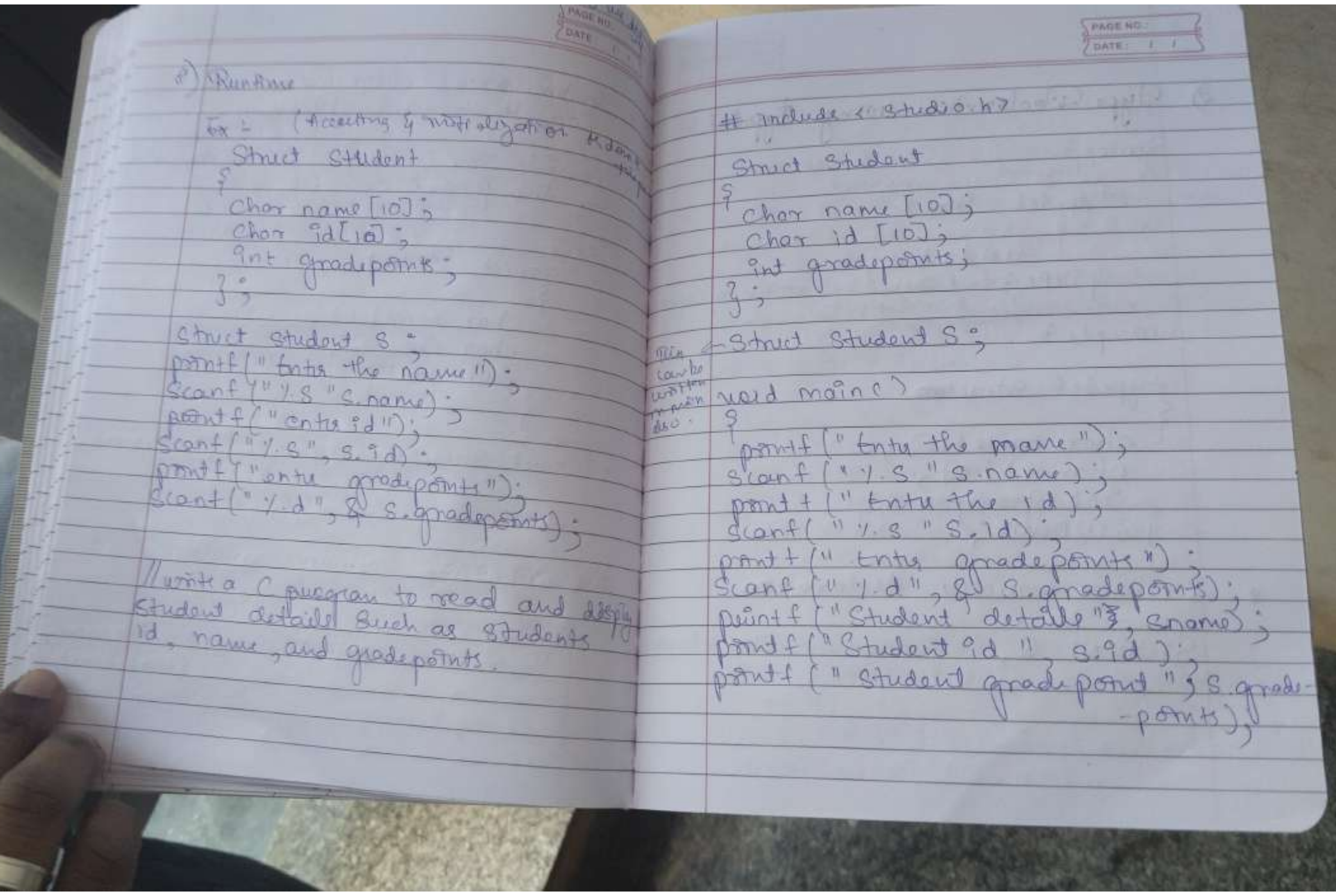
* Accessing structure

-> S . name , S . id , S . gradepoints .

* Multiple structure variable declaration.

STUDENT . S1, S2, S3 ;

```
// write a C program to multiple
two fraction number using structure

# include <stdio. h>

typedef struct
{
  int numerator ;
  int denominator ;
} FRACTION ;

void main ()
{
  FRACTION fr1, fr2, res .
printf ("Enter first fraction in the form
  of x/y ");
Scanf ("%d%d ", & fr1. numerator, & fr1.
  denominator) ;
```



PAGE NO.:
DATE: / /

8) Runtime

Ex :- (Accecting & initializ ation of R.data [illegible]

```
 Struct Student
 {
  Char name [10];
  Char id[10];
  int gradepoints;
 };

Struct student S;
printf("Enter the name");
Scanf("%S" S.name);
print f("enter id");
Scanf("%S", S.id);
print f("enter gradepoints");
Scanf("%d", & S.gradepoints);
```

//write a C program to read and display Student details such as Students id, name, and gradepoints.

PAGE NO.:
DATE: / /

```
# include <studio.h>

 Struct Student
 {
  Char name [10];
  Char id [10];
  int gradepoints;
 };

 Struct Student S;

void main()
{
 printf("Enter the name");
 Scanf("%S" S.name);
 print f("Enter the id);
 Scanf("%S" S.id);
 print f("Enter gradepoints");
 Scanf("%d", & S.gradepoints);
 print f("Student details"}, Sname);
 print f("Student id", S.id);
 print f("Student gradepoint"} S.grade-
                                 -points);
```

This can be written in main also.

Example :

```
Struct Student
{
  Char name [10];
  char id[10];
  int gradepoints;
};
```

when we ~~declare~~ the memory is not allocated but to declare also memory we need declare the variable Structure.

2) ~~Type Declaration~~

* Syntax To declare Structure variable

Syntax : To declare Structure variable

Struct Structurename variablename;

Ex ; Struct Student S ;

This will create the variable outside to declare the variable inside the only you can add S to

S

10 for

10

9

id gradepoints

10+
10+
9
29

allocated to Structure S.

* Initialization : Two types
1) Compile time 2) Run time (inputs taken at run time

1) Compile time :- we can initializing value in the program

Ex :

```
Struct Student.
{
  char name[10];
  char id [10];
  int gradepts;
};
```

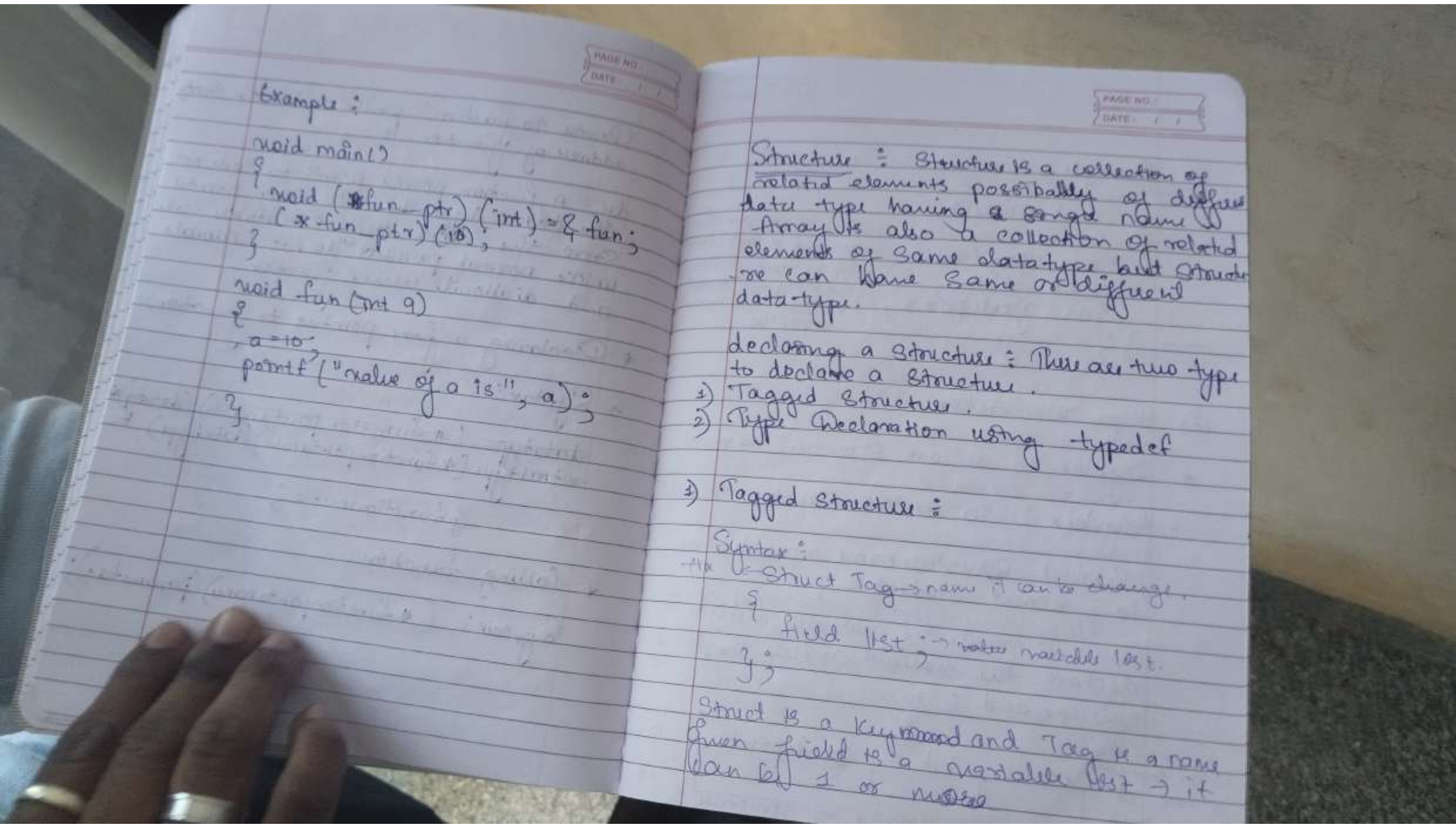
initialization

```
Struct Student S = {"xyz", "29119C9401",
                    9};
```

3) * Accessing Structure variable:-

Syntax :

Structure variable name.variablename;
(member name)



Example :

```
void main()
{
  void (*fun_ptr)(int) = & fun;
  (*fun_ptr)(10);
}

void fun(int a)
{
 ~~a=10;~~
printf("value of a is", a);
}
```

Structure : Structure is a collection of related elements possibally of differe data type having a single name. Array is also a collection of related elements of same datatype but structure we can have same or different data-type.

declaring a structure : There are two type to declare a structure.

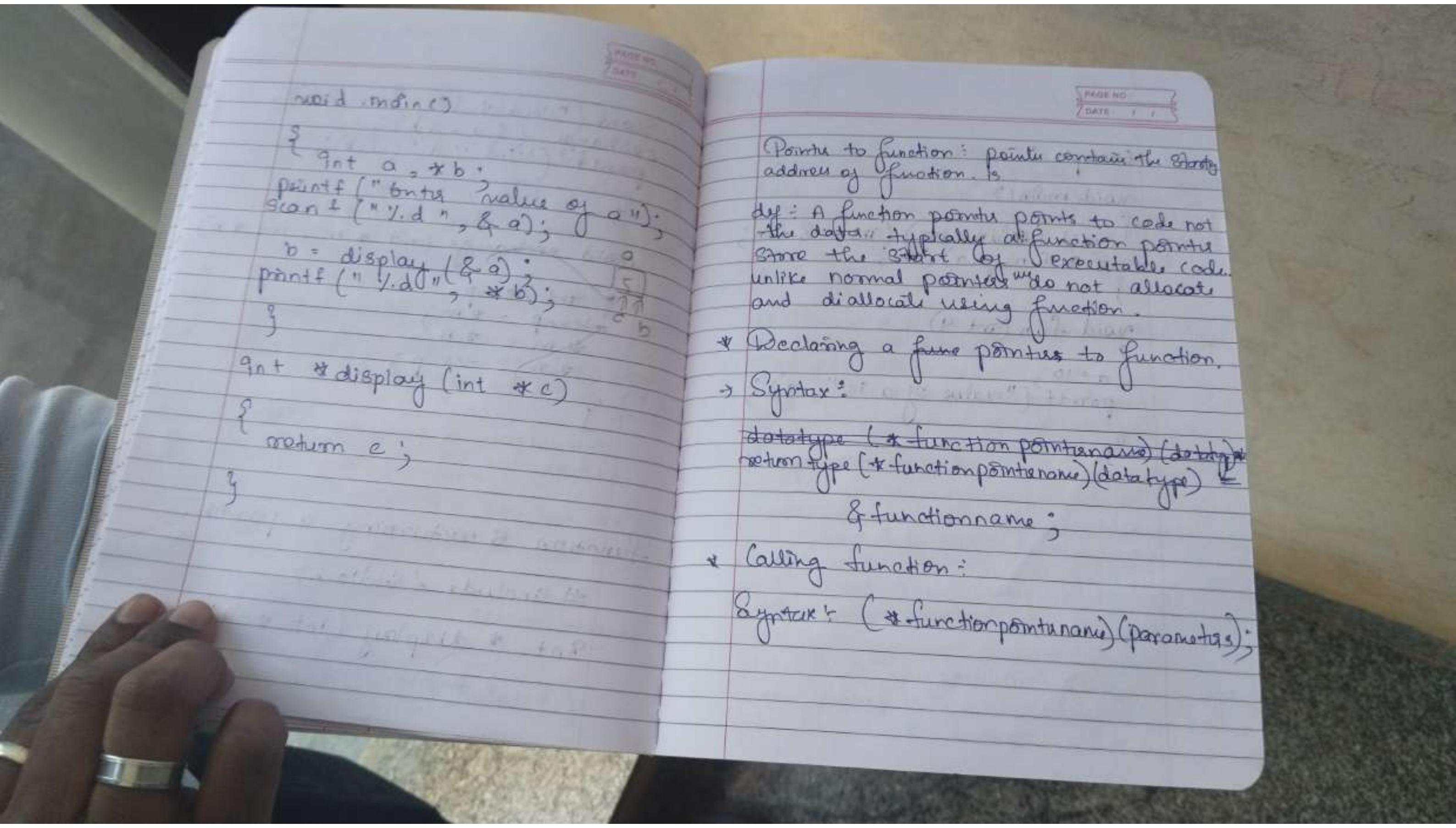
1) Tagged Structure.
2) Type Declaration using typedef

1) Tagged structure :

Syntax :

```
Struct Tag → name it can be change.
{
    field list; → value variable list.
};
```

Struct is a Keyword and Tag is a name given field is a variable list → it can be 1 or more



```
void main()
{
   int a, *b;
printf("Enter value of a");
scan f("%d", &a);

   b = display(&a);
printnf("%d", *b);
}

int *display(int *c)
{
  return c;
}
```

Pointer to function: pointer contain the starting address of function. is

def: A function pointer points to code not the data. typically a function pointer store the start of executable code. unlike normal pointers we do not allocate and diallocate using function.

* Declaring a ~~func~~ pointers to function.

→ Syntax:

~~datatype (*function pointername) (datatype)~~

return type (*functionpointername)(datatype) = &functionname;

* Calling function:

Syntax: (*functionpointername)(parameters);

Pointers and functions

This concept is similar to pass by address and pass by value. pointers can be passed to a function and can be return to a function.

1) Passing pointers to a function. [pass by address].

Eg : Swapping the content of two variables using pointers.

```
# include <Stdio.h>
void exchange (int *m, int *n);
void main()
{
 int a, b
printf ("Enter a & b values");
```

declare function   function name

two parameter because

```
Scanf ("%d %d", &a, &b);
exchange (&a, &b);
printf ("%d %d", a, b);
}
```

we need to pass the address of a, b

calling function

a 5 b

called function or implementation of function

```
void exchange (int * m, int * n)
{ int temp
  temp = *m;
  * m = * n;
  * n = temp;
}
```

temp = 5

= 10

a [5] b [10]

↓ m ↓ n.

In the memory we are exchange a & b values.

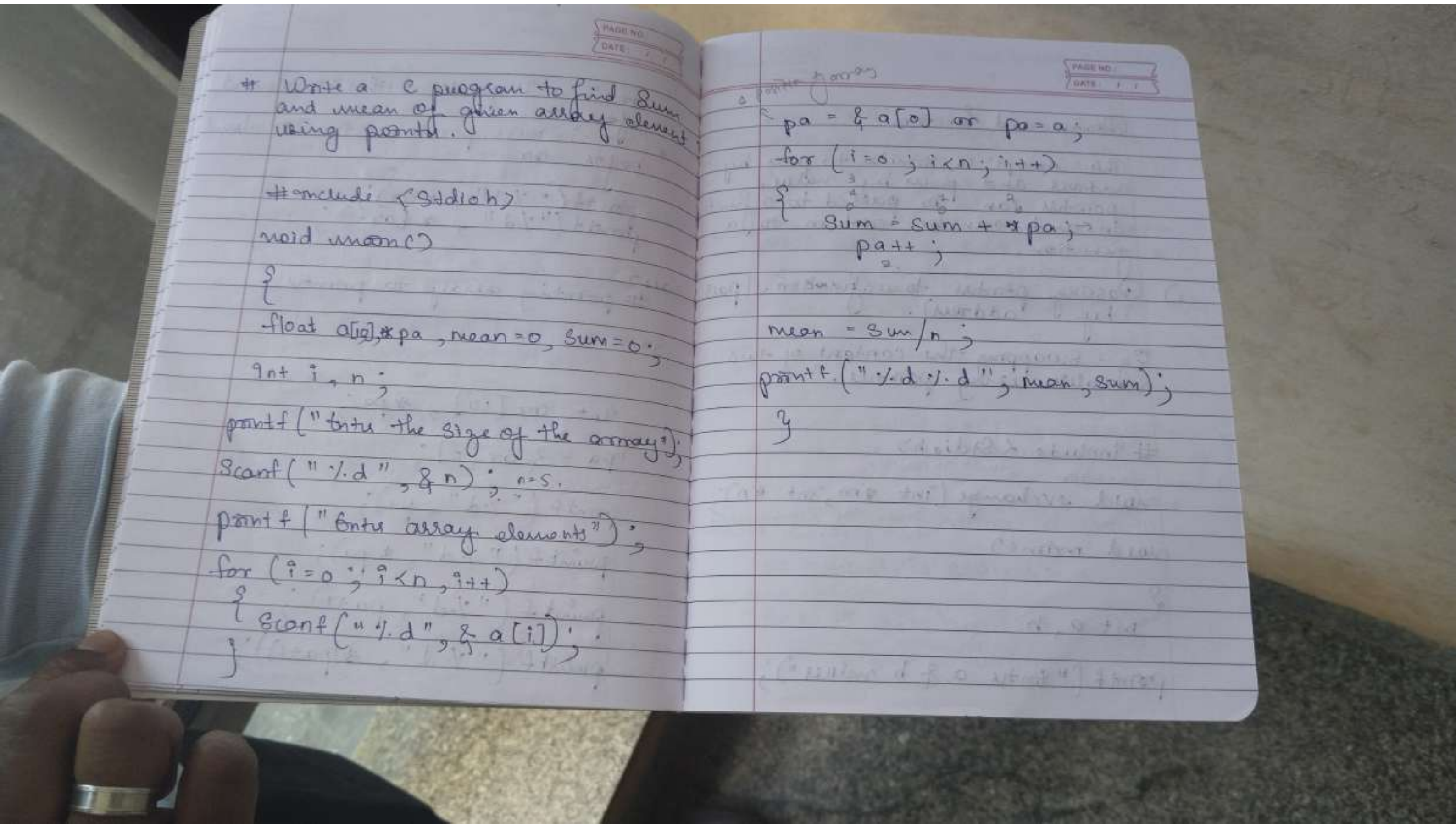
This program is pass by address by using address we are changing the values of

function & returning a pointer.

```
# include <Stdio.h>
int * display (int *c);
```

declaring function

* because function is going to return value.



Write a C program to find Sum and mean of given array elements using pointer.

```
#include <stdio.h>
void main()
{
float a[10],*pa, mean=0, sum=0;
int i, n;
printf("Enter the size of the array");
scanf("%d", &n); n=5.
printf("Enter array elements");
for (i=0; i<n, i++)
{ scanf("%d", & a[i]);
}
```

a pointer array

```
pa = & a[0] or pa=a;
for (i=0; i<n; i++)
{
    sum = sum + *pa;
    pa++;

mean = sum/n;
printf("%d %d", mean, sum);
}
```

Syntax of declaring ~~an~~ array
datatype arrayname[size]
int ar[10];

Pointers and arrays.

we can use arrayname to point as a pointer

1 every array is a pointer
2 value of array variable is equal to address of first element.
3 elements of the array can be access using pointers.

ar	ar+1	ar+2					
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7 → max

1234 1234
address of 1 element
2 element

ar-i → will give

case 1 Program.
* Array name as a pointer.

```
int ar [10]; → declaring array
initializing value not done

printf ("%.d", ar[0]);  1
printf ("%.d", &ar[0]);  1234
printf ("%.d", ar);  1234 address of
```

* ar will print value of 1 element.

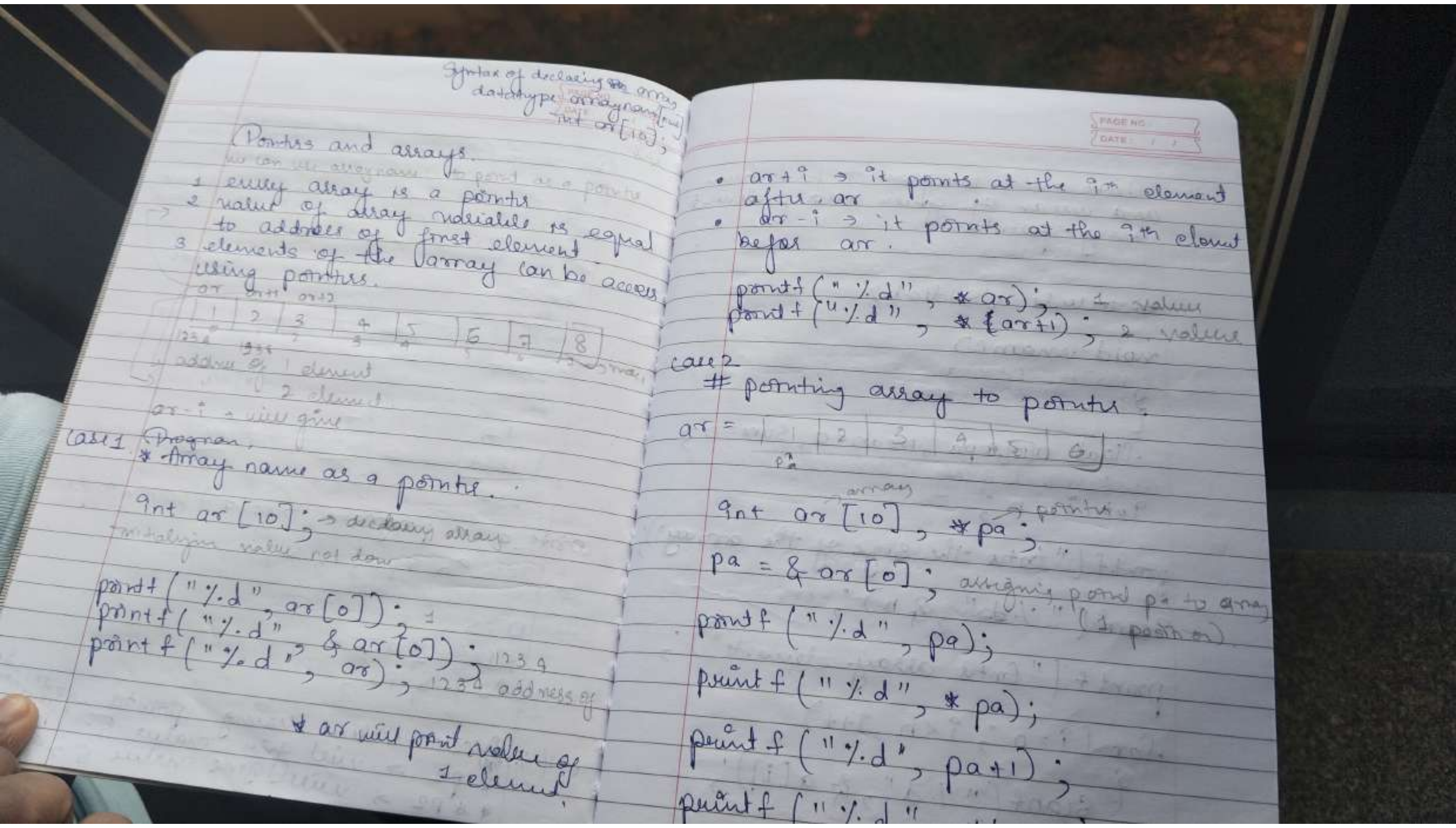
- ar+i → it points at the ith element after ar.
- ar-i → it points at the ith element befor ar.

```
printf ("%.d", *ar);  1 value
printf ("%.d", *(ar+1));  2 value
```

case 2
# pointing array to pointer.

ar =	1	2	3	4	5	6
	pa					

```
int ar[10], *pa;
      array   pointer
pa = &ar[0];  assigning point pa to array (1 position)
printf ("%.d", pa);
printf ("%.d", *pa);
printf ("%.d", pa+1);
printf ("%.d", *(pa+1));
                 pa++
```



Syntax of declaring ~~the~~ array
datatype arrayname[size]
int ar[10];

Pointers and arrays.

we can use arrayname to point as a pointer

1 every array is a pointer
2 value of array variable is equal to address of first element
3 elements of the array can be access using pointers.

ar ar+1 ar+2

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

1234 address of 1 element
1234 2 element

ar - i → will give

case1 Program.
* Array name as a pointer.

```
int ar [10]; → declaring array
initializing value not done

printf ("%d", ar[0]); 1
printf ("%d", &ar[0]); 1234
printf ("%d", ar); 1234 address of
```

* ar will print value of 1 element

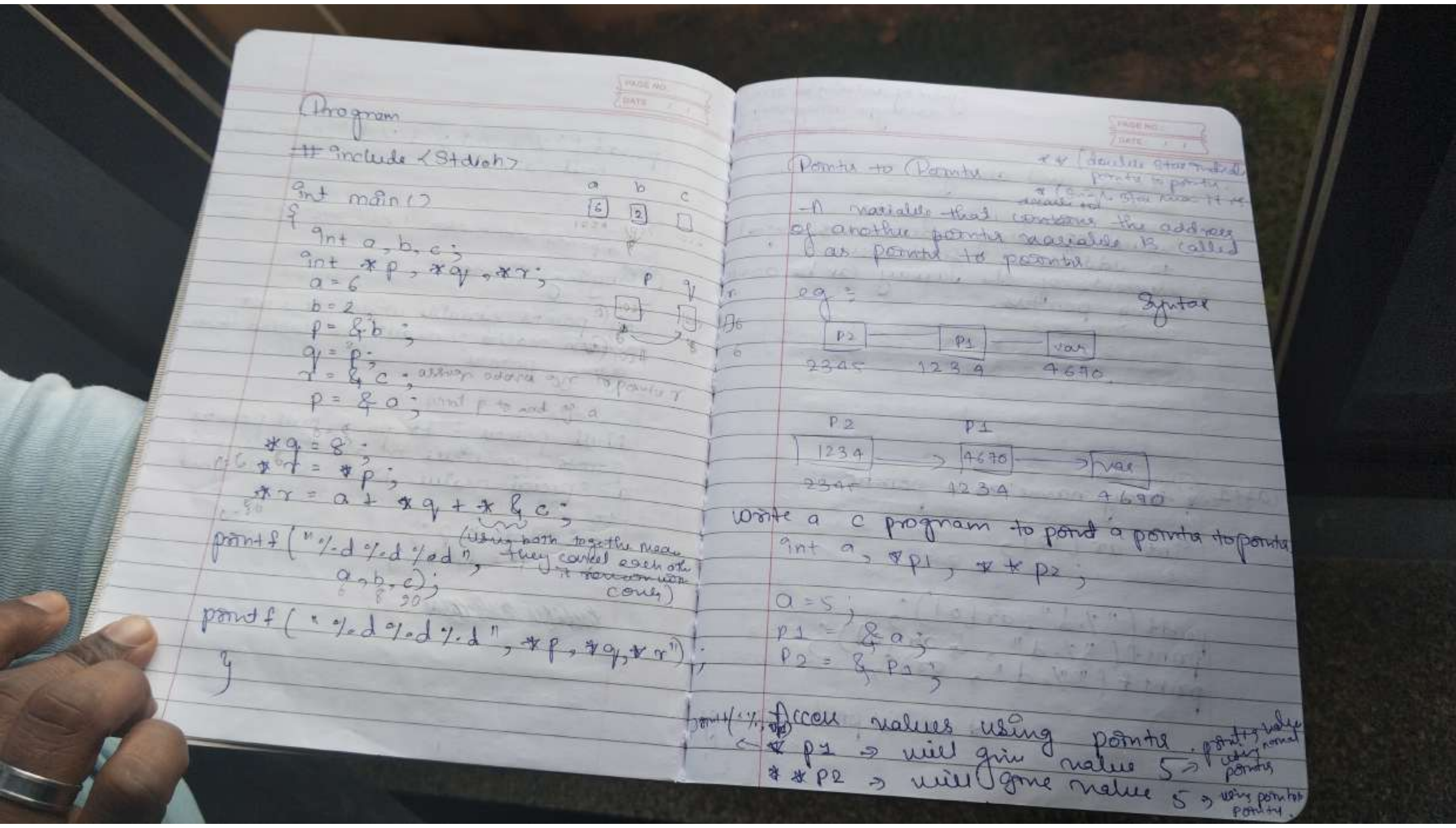
- ar+i → it points at the ith element after ar
- ar-i → it points at the ith element befor ar.

```
printf ("%d", *ar); 1 value
printf ("%d", *(ar+1)); 2 value
```

case 2
# pointing array to pointer.

ar = | 1 | 2 | 3 | 4 | 5 | 6 |
pa

```
int ar [10], *pa;   (array) (pointer)
pa = & ar [0]; assigning point pa to array (1 position)
printf ("%d", pa);
print f ("%d", * pa);
print f ("%d", pa+1);
printf ("%d"
```



Program

```
#include <Stdio.h>

int main()
{
  int a, b, c;
  int *p, *q, *r;
  a = 6
  b = 2
  p = &b;
  q = p;
  r = &c; assign address of r to pointer r
  p = &a; point p to add of a

  *q = 8;
  *r = *p;
  *r = a + *q + *&c;
printf("%d %d %d", a, b, c);
printf("%d %d %d", *p, *q, *r");
}
```

(using both together mean they cancel each other it [illegible] conv)

a b c
6 2

p q r

Pointer to Pointer

** (double star indicates pointer to pointer)

* (single star [illegible] it is [illegible])

-A variable that contains the address of another pointer variable is called as pointer to pointer.

eg :- Syntax

p2 — p1 — var
2345 1234 4670

p2 p1
1234 → 4670 → var
2345 1234 4670

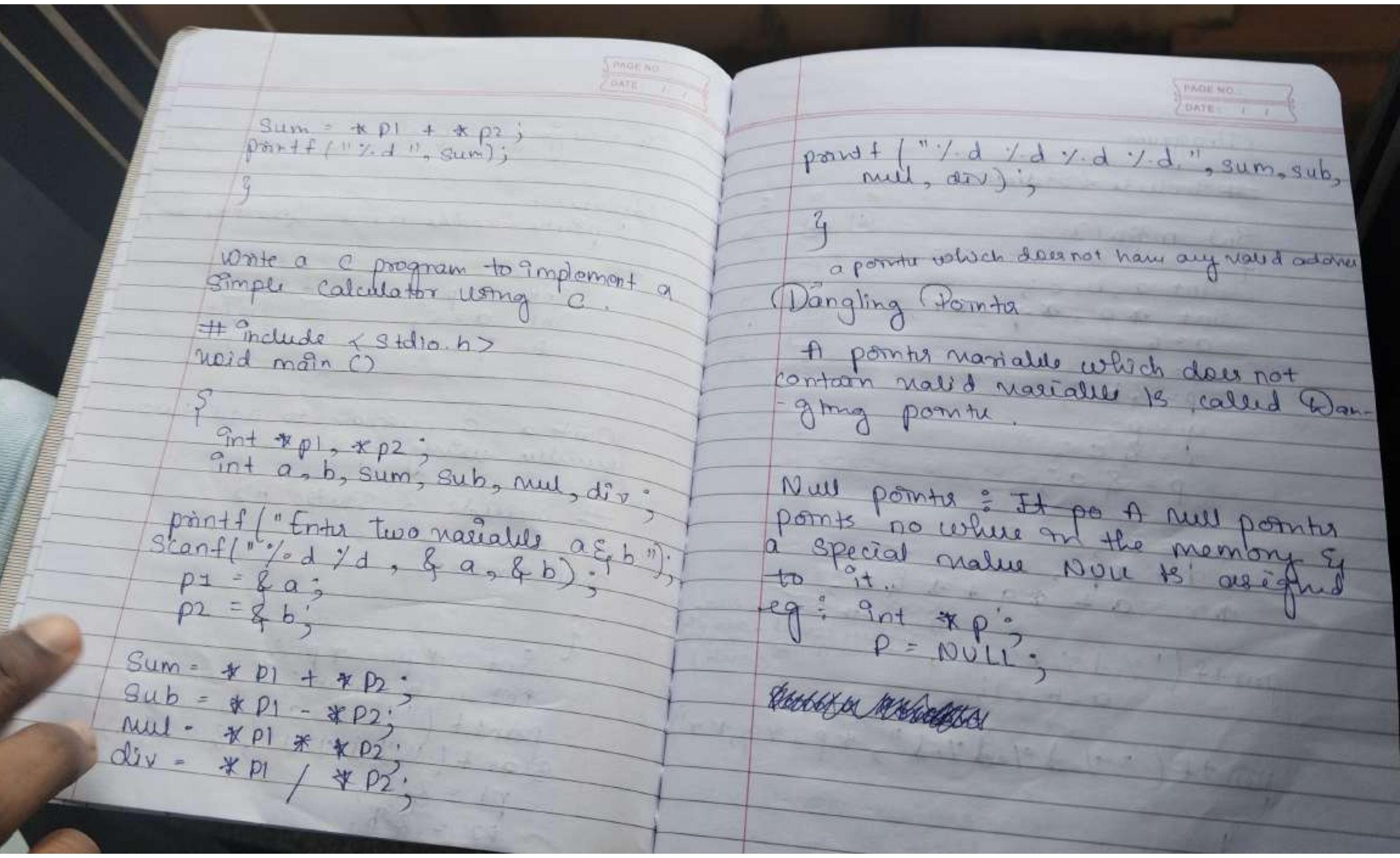
Write a C program to print a pointer to pointer

```
int a, *p1, **p2;

a = 5;
p1 = &a;
p2 = &p1;
```

Access values using pointer.

*p1 ⇒ will give value 5 ⇒ printing value using normal pointer

**p2 ⇒ will give value 5 ⇒ using pointer to pointer.



PAGE NO.
DATE / /

```
Sum = *p1 + *p2;
printf("%d", sum);

}
```

Write a C program to implement a simple calculator using C.

```
# include <stdio.h>
void main ()
{
  int *p1, *p2;
  int a, b, sum, sub, mul, div;
 printf("Enter two variable a & b");
Scanf("%d %d, & a, & b);
 p1 = & a;
 p2 = & b;

Sum = *p1 + *p2;
Sub = *p1 - *p2;
mul = *p1 * *p2;
div = *p1 / *p2;
```

PAGE NO.
DATE / /

```
printf("%d %d %d %d", sum, sub,
    mul, div);

}
```

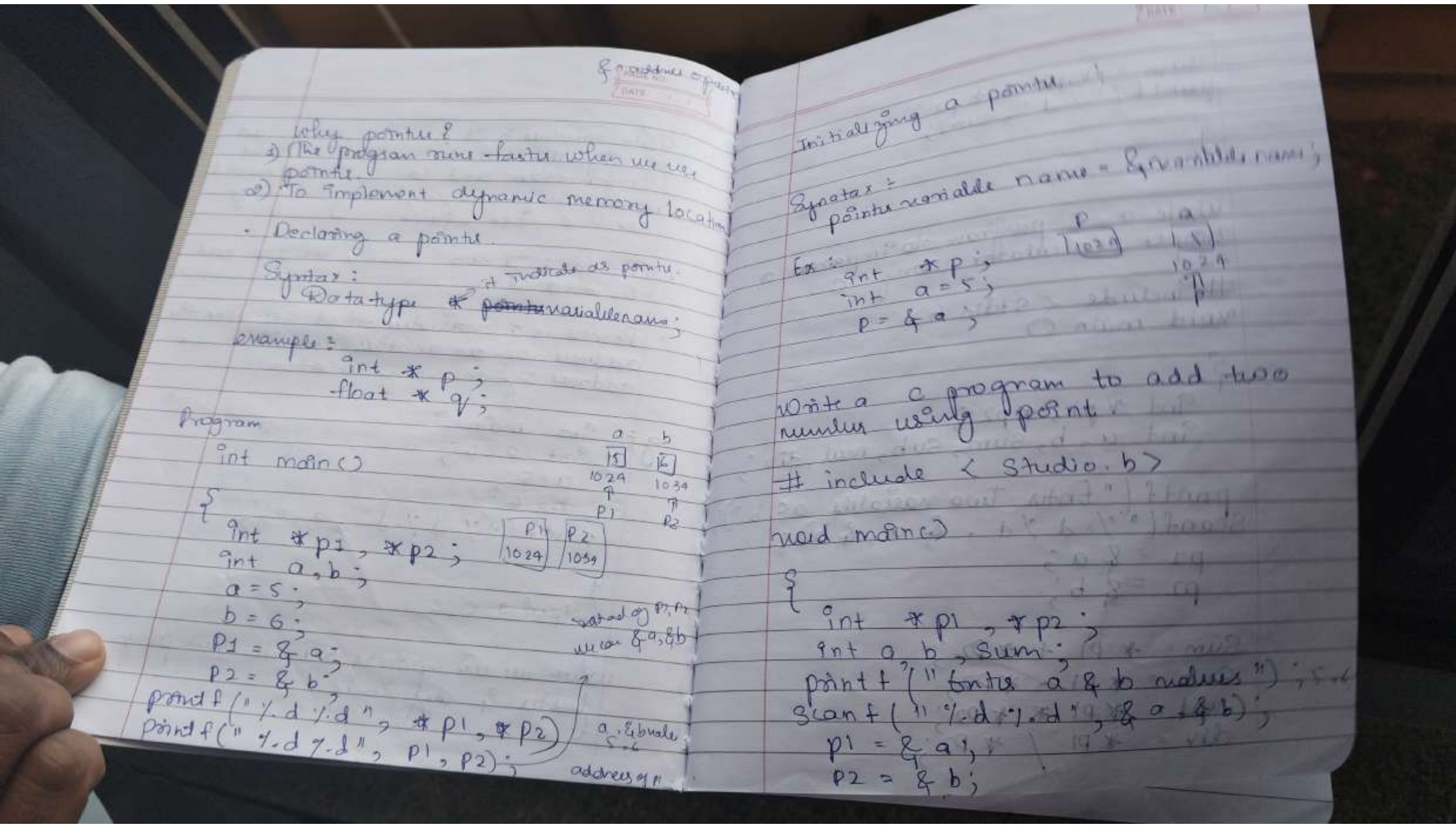
a pointer which does not have any valid address

(Dangling Pointer

A pointer variable which does not contain valid variable is called Dangling pointer.

Null pointer : ~~It po~~ A null pointer points no where in the memory & a special value NULL is assigned to it.

eg : int *p;
 P = NULL;



& is address operato

Why pointer?

1) The program runs faster when we use pointer.

2) To implement dynamic memory location

- Declaring a pointer.

Syntax:

it indicate as pointer.

```
Data-type * ~~pointer~~variablename;
```

example:

```
int * p;
float * q;
```

Program

```
int main()
{
  int *p1, *p2;
  int a, b;
  a = 5;
  b = 6;
  P1 = & a;
  P2 = & b;
printf("%d %d", *p1, *p2)
print f("%d %d", p1, p2);
```

a	b
5	6
1024	1034
P1	P2

P1	P2
1024	1034

instead of P1, P2 we can &a, &b

a, b values 5, 6

address of P1

Initializing a pointer

Syntax:

```
pointer variable name = &variable name;
```

Ex:

```
int *p;
int a = 5;
p = & a;
```

p	a
1024	5
	1024
	p

Write a C program to add two number using point.

```
# include < Studio. h>

void main()

{
  int *p1, *p2;
  int a, b, sum;
  printf("Enter a & b values"); 5 6
  scanf("%d %d", &a, &b);
  p1 = & a;
  p2 = & b;
```

Data structure : It is a concepts of set of algorithms used to structure the information and can be implemented using any programming language like C, C++, java.

- Pointers : It is a variable used to hold the address

- Pointer is a variable that contains the address of ~~at~~ another variable or address of memory location.

eg :
```
int main()
{ int a, b;
  a = 5;
  b = 6;                format specific for integer.
  printf("%d %d", a, b);
}
```
output → 5, 6.

When we use pointers are to run program
faster & dynamic