— jenny lectures (videos)

C language -

[close to human] (variables, loops & all)

low level vs High level language

- → machine language [bits only to understand m/c]
- → Assembly language [memonics used symbols & numbers]

Hardware (1) → machine lang. (2) → Assembly lang (3) → High level lange (4) → web/S/W (applications) (5)

Complier → complete source code convert machine lang.

HLL → [complier] → m/c code

interprete → Reads only line by line & execute &

HLL → [interpreter] → m/c code

* features of c language.

1. High language
2. small language having 32 keywords.
3. portable program.
4. built in function & operators
5. Structure language.
6. pointers ⇒ store the add.
7. Dynamic memory allocation

* Structure of C program.

① **Documentation section**

// ⇒ cmd line

/* ——— */ (multiple line cmd)

② **link section**

header file ext

header, #include <stdio.h>

↓ standard i/o.

pre processor

printf (o/p)

scanf (i/p)

console i/p/o/p

[getch()] ⇐ [conio.h] ⇒

Pre defined func

③ **Def^n section** (micro defination)

↳ # define PI 3.14159

④ **Global defination section.**

(variables used more than 2 fun then define that out full code)

⑤ **main section** (Only 1 fun)

⑥ **Sub program section**

⑤ main fu^n (Only 1 time)

→ void main()

{

Declaration (optional);

Executable (at least one); (compulsory)

}

* Constants in C (const)

↳ (fix values)

- Numeric
  - Integer
    - → Decimal (0-9)
    - → Octal (0-7)
    - → Hexa.d (0-15)
  - Floating (Real constant) (25.2)
- Character
  - Single char. ('5' 'A' ',' '@') → [ascii value]
  - String constant ["Hello"]

* Variables in C

Declaration

[Datatype] [Variable names]

→ int a;

Rules

* Executation process in C

source code (first.c)

↓

[Preprocessor]

↓ first.i

[compile]

↓

Syntax error → Yes → source code

NO ↓ first.asm

[Assembler]

↓ first.obj (0,1,0)

[link with system lib.]

↓ first.exe

[execute] ← [i/p data]

↓

logic / dat & error → Yes → [i/p data]

NO ↓

[o/p]

Scanned with CamScanner

x--	post
--x	pre

* Identifiers, & keywords in C

(always in small letter)

keywords ⟹ 32 total (32)

[int, float, break, if, else, while, do, auto, double,]
{predefined word}

31 chara.

[ex. Sum_2 ✓
2Sum ✗]

* Data types in C

↳ (how much memory to allocated to variable)

- Primary
  - int
    - [2 bytes]
    - short int (2 byte) (integer value) Range [-32768 to 32767] 16bit
    - long int (4 byte)
  - float
    - (4 byte) [3.45 to 3.4e38] Float a
    - double → 8 byte
    - long double → 10 bytes
  - char
    - char a
    - [1 byte = 8 bit]
    - signed [-128 to 127]
    - unsigned [0 to 255]
  - double
- Desired
  - Array
  - structure
  - Union
  - pointers
  - void (empty)
- User-defined
  - type-def
  - typedef

* Operators in C.

(Types of operator based on operand)

① Unary
- — unary minus ex- (-b)
- ++ (incre) — (decr.)
- !! (Not logical not)
- & (add. of operators)
- sized of

② Binary
- Arithmetic (+, -, *, /, %)
- Relational (<, >, <=, >=)
- logic (&&, ||)
- Bitwise (&, |, <<, >>, ~, ^)
- comma oper. (,)
- Assign. ope. (=)

③ Tenary
- ? : (ternary)

ex1 ? ex2 : ex3
true

(% = only used int value)

(14)

Remender

① Arithmetic operators (+, -, *, /, %)

Higher prec.
* / % — ①
+ - — ②
L → R

```
#include <stdio.h>
#include <conio.h>   → old version
void main()
{
    int a = 10, b = 7;
    clrscr();
    printf("a+b = %d", a+b);   = 17
    printf("a-b = %d", a-b);   = 3
    printf("a*b = %d", a*b);   = 70
    printf("a/b = %d", a/b);   = 1
    printf("a%b = %d", a%b);   = 3
    getch();
}
```

(Format specifier for int.)

H.W ⟹ Int a = 10, Float b = 7

(15)

* prefix ⟹ ++x, --x

↳ (first value incr. & used)

postfix = x++, x--

↳ (first used value then incr.)

y = x--;
⇓ also write
y = x;
x = x - 1;

```
#include <stdio.h>
void main()
{ int x = 10, y;
    y = ++x;
    printf("%d", x);
    printf("%d", y);
}
```

y = --x;
⇓ also write
y = x - 1;
y = x;

Scanned with CamScanner

True = 1
False = 0

(15)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a = 5, b, c, d
```

a	b	c	d
5			

a=5 b c d

6 6 . . ⇐ b = ++a; → pre incr. ⇒ value incr & updatable & value used

6 6 6 . ⇐ c = a++; → post incr. 6 original value used

8 6 6 8 ⇐ d = ++a;

```
printf("%d", a);
}
```

(16)

* Relational – (compair to two expression)

[< , > , <= , >= , == , !=]

less than, greater than

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a=18, b=9;
clrscr();
printf("%d", a<b);
printf("%d", 'c' & 'b');
getch();
}
```

a	b
8	9

int, less than

ASCII 99 98

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a=18, b=9, c, d, e=10;
clrscr();
```

a	b	c	d	e
18	9	9	10	10

update used 10

c = b++; (post incr.) → c = 9 & b = 10 update

d = b; → d = 10

L → R

a<b = 18<10 = 0 ⇐ printf("%d", a<b<c>d); = 0

18<10 10<9 1>10 0

b<c = 0<9 = 1

c>d = 1>10 = 0

Final

printf("%d", b==e); = 1

(10==10) =

c = 9+1 = 10 > 10 = 0

printf("%d", c+1 > e); = 0

Ar. 1 Re. L → R

printf("%d", a+c==b>e<c+d);

18+9 = 27 == 10>10<19

0<19

27 == 1 → 0

getch();

}

New

f = a>b>d<c; = 1

printf("%d", f1 = 1); = 0

27 == 1 != 1

0! = 1

printf("%d", a+c==b>=e<c+d !=1); = 1

18+9 = 27 == 1 != 1 10<19 9>=1 = 1 1!=1 = 1

(17)

* Logical (&& || !) — (more than one cond^n checking)

AND OR NOT

$\boxed{a = 10}$ $\boxed{b = 5}$

&& AND = (Both operat. is true = 1)

First is false 0 then not check 2nd expression

10 > 5   5 != 10   5 < 10   10 > 5

a > b && b != 10 && b < 10 && a > 5

1   1   1   1

a < 5

0

= 1 (True)   0 (false)

OR = (||)

$\boxed{a = 10}$ $\boxed{b = 5}$

Only operate is true = 1 then not check 2nd expression

10 < 5   5 == 4

a < b || b == 4   (o/p)

0   0   = 0

0	0	0
1	0	1
0	1	1
1	1	1

10 > 5   5 == 4

a > b || b == 4   (o/p)

1 || 0   = 1

NOT = (!) ⇒ (compliment of expression / Negation of the value)

o/p

!5 = 0   (!1 = 0)

!0 = 1   (!0)

10 > 5

!(a > b)

1

!1 = 0 o/p

---

a   b

$\boxed{1}$ $\boxed{6}$

0   (7)

a-- && ++b;

1   1   1+6

Result1 = 1

Result2 = 0

--a && ++b;

0   1

void main ()

{

int a = 4, b = 6, result;

4 > 6 = 0 ← Not

1st expression 0 then o/p is 0

result = a > b && printf ("Java");

a < b = 1 && 1

printf ("%d", result); = 1

o/p Java1

}

result = a > b && printf ("Java")

4 > 6 = 0   x = 0

1st = 0 then check Next

check = 0

|| printf ("Gadi");

1

printf ("%d", result); = 1

Gadi 1

#

{ int a = 4, b = 6, result;

a	b	result
10	5	
11		

printf ("%d", 4 && 10); o/p

1st

1 = 1 = (1)

{

int a = 10, b = 5, result;

result = (a > b) && a++;

10 > 5 || 10

1 && 1 = 1 (o/p)

printf ("%d", result); = 1

printf ("%d", 4 && 0); = (0)

0

printf ("%d", a); = 11

int a = 1 b = 6

result = a-- && ++b;

result = --a && ++b;

H.W

(18)

* Bitwise

~~& | ^~~

1 byte = 8 bits

& = AND
| = OR
^ = XOR
~ = Not
<< = left shift
>> = Right shift

a = 10, b = 5

c = a & b

= 10 & 5

Binary 1010 0101

c = 0

```
  1010
& 0101
-------
  0000
    ↓
    0
```

| = OR

a = 10, b = 5

a | b

15

```
  1010
| 0101
-------
  1111
```

XOR		
0	0	0
0	1	1
1	0	1
1	1	0

^ = XOR

a ^ b

15

```
  1010
^ 0101
-------
  1111
```

~ ~~NOT~~

```
int a=10, b=6;
printf("%d", a&b);
printf("%d", a|b);
printf("%d", a^b);
printf("%d", a&b & b+1 || 0);
}
```

10 & 7   6+1=7

2   48   7

1110 = 1

(19)

* Left shift <<

multi

variable << No.

short trick

int a = 10;

c = a << 2;

10 × 2^2

10 × 4

= 40

c = 40

0	0	0	0	1	0	1	0

left shift

0	0	1	0	1	0	0	0

New

40

- Right shift >>

division

a = 10

c = a >> 2

10 / 2^2

= 10/4 = 2

c = 2

0	0	0	0	1	0	1	0

Right

0	0	0	0	0	0	1	0

New

2

- ~ NOT

int a = 5;

b = ~a;

b = 10

Binary

0 1 0 1

~

1 0 1 0

short cut

a = -(a+1)

int a = 5

① (a+1) << 2

6 << 2 — left shift

6 × 2^2 = 6 × 4 = 24

1000

② 5+3 ③

		o/p
①	(a+1) << 2	24
②	(a+2) >> 1	8
③	~(a+8)	7.

② (a+2) >> 1

(5+2) >> 1

7 >> 1

③ ~~7/2~~

0	0	0	0	0	1	1	1

1

(20) 22 (3) 25

* Special operators

(, , size of , pointer * , & , . , →)

comma

Seperatable

```
int a;
a = (printf("Java"), 2);
```

a = 2

Print Java

2

```
int a;
a = (5, 4);
```

a = 4

```
int a;
a = 5, 4
```

a = 5

```
int a = (5, 4);
```

a = 4

```
int a = 5, 4;
```

error

```
int a = 8, b;
b = (a++, ++a);
```

a	b
8	
9	10
10	

a	b
8	8
9	10

```
b = a++, ++a;
```

(21) *

operators	Order	Priority	
( ) . → ++ -- (postfix)	L→R	1	
++ -- + - ! ~ * & size of (prefix)	R→L	2	Unitary
* / %	L→R	3	Binary
+ -	L→R	4	
<< >>	L→R	5	
<<= >>=	L→R	6	
== !=	L→R	7	Equality op.
&	L→R	8	Bit
^	L→R	9	
|	L→R	10	
&&	L→R	11	
||	L→R	12	
?:	R→L	13	
= += -= /= %= *= ^= >>= <<=	R→L	14	Assignment
,		15	

(22) *

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int a, b, sum;
  clrscr();
 printf("Enter two nu.");
 scanf("%d %d", &a, &b);
   sum = a + b;
 printf(" sum is = %d ", sum);
  getch();
}
```

Run - check errors &

(23) * Formatted input fun in C

scanf() - [i/p from user]

("control string", argument eg.1, eg.2 ...);

%d &a &b

add. of a add of b

```
#
#include <stdio.h>
#include <conio.h>
void main()
{
int a=5, b=4, sum;
clrscr();
printf("Enter two no.");
scanf("%d %d",
        &a, &b);
sum = a+b;
printf("sum = %d", sum);
getch();
}
```

```
int a, float b, char c;
printf("Enter integer value, float value,
        character");
scanf("%d %f %c", &a, &b, &c);

printf("interger = %d, character = %c, float is = %f",
        a, b, c);
getch();
```

Scanned with CamScanner

(24)

* printf() ⇒

printf("control string", arg1, arg2 ... argn);
(Name of variable direct)

① print simple text
↳ printf("~~Enter~~ Hello"); ⇒ Hello

② format specifier
printf("%d", sum); ⇒ Results.

③ width of o/p a = 1234
(Right) printf("%6d", a);

			1	2	3	4

④ (Left) printf("%-6d", a);

1	2	3	4		

⑤ Float printf("%10.2f", a);
a = 1234.5678

			1	2	3	4	.	5	6

(25)

* unformatted i/p ~~for~~ fun in c.

getchar() — (single only chara.) → char ch; ch = getchar();
getch() — (Hold the screen) [ch = getche();] printf("%c", ch);
getche()
o/p [ch = J] o/p key press J
gets()

```
{
char ch[10];
gets(ch);
printf("ch = %s", ch);
}
```

array, string

Enter ⇒ Hellow — (Enter)
ch = Hellow

(26)

* unformatted o/p fun in c -

- putchar()] single char o/p
- putch()]
- puts()] full string.

header → #include <stdio.h>
header → ~~#include <conio.h>~~

printf("char = %c", ch);
⇓ - OR -
putchar(ch); [single char]
same ⇓
putch(ch);

putchar() ⇒ putchar(variable Name)

puts() ⇒ (string)

```
void main()
{ char ch[20];
  puts("Enter a string");
  ~~getch~~ gets(ch);
  puts(ch);
```

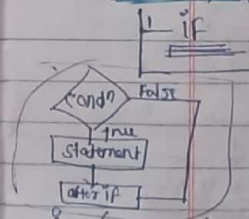
⇒ enter a string :
Hello
→ Hello,

puts('a'); ⇒ o/p

(27)

* Control statements in c

1. if
2. if else
3. if else if ladder
4. Nested if
5. switch



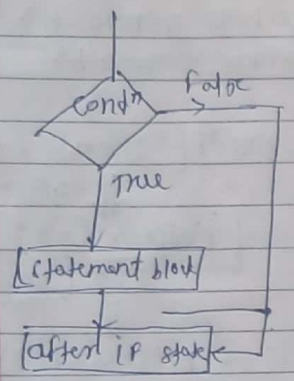
if (cond^n)
Statement,
state if statement

```
void main()
{ int a, b, sum;
  printf("Enter a & b");
  scanf("%d %d", &a, &b);
  sum = a+b;
  printf("sum = %d", sum);
  ~~getch()~~;
}
```

```
void main()
{
int a
printf("Enter a");
scanf("%d", &a);
if(a) printf("Inside the");
printf("outside the");
getch();
}
```

a = 5 ⇒
Inside the

printf("Dont want to calculate sum");
getch();




```
void main()
{ int age;
  printf("Enter age : ");
  scanf("%d", &age);
if (age > 20)
  {
    printf("your age is %d", age);
    printf("you can go coffee with me");
  }
  printf("Its time to go home");
  getch();
}
```

(o/p) →

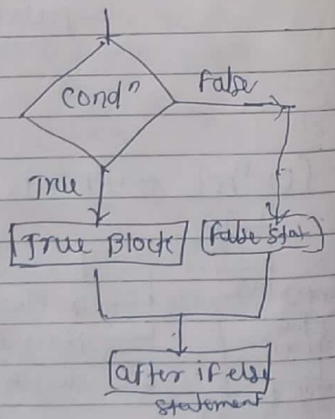
Enter age = 21
your age is 21
you can go coffee with me

Enter age = 15
its time to go home

(28)

* if_else

```
if (cond^n)
  {
    (True block state.)
  }
else
  {
    (False block state.)
  }
after if else block statement.
```



(o/p)

Enter age = 25 / 28
age = 28
coffee with me
= out of if else
Enter age = 20
age = 20
Go home
⇒ out of if else

```
Void main()
{ int age;
  printf("Enter age");
  scanf("%d", &age);
  20 > 25 = 0 (exit)
if (age > 25 && age < 30)
  {
    28 > 25 = 1   28 < 30 = 1
    printf("age = %d", age);
    printf("Coffee with me");
  }
else
  {
    printf("age = %d", age);
    printf("Go home");
  }
printf("out of et if-else");
getch();
}
```

(29)

* Nested if

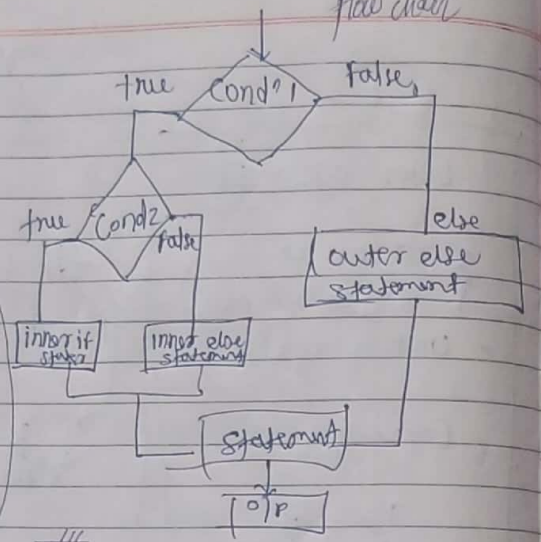
```
if (cond^n1)
  {
    if (cond^n2)
      {
        if (cond^n3)
          {
            (Inner if Block);
          }
        else
          {
            (inner else block);
          }
      }
```

(29)

* Nested if

```
{
  if (cond^n 1)  (true)  (false)
  {
    if (cond^n 2)  (check)
    {
      Inner if state ;
    }
    else
    { Inner else state ;
    }
  }
  else
    { outer else state ;
    }
    (chuk)
  Statement x ;
}
```

Flow chart




```
void main()
{ int age, salary;
  printf("Enter age & salary");
  scanf("%d %d", &age, &salary);
  53 > 50 = 1.
  if ( age > 50)
  45 > 50 = 0 greater than
  {
    if (salary < 60k)
    {  45k < 60k = 1
       70k < 60k = 0
       salary = salary + 10k;
    }
    else { salary = salary + 5k;
    }
  }
  else
  { salary = salary + 3k;
  }
  printf(" End of program");
  getch();
}
```

O/P

Enter age = 53
salary = 45k / 70k

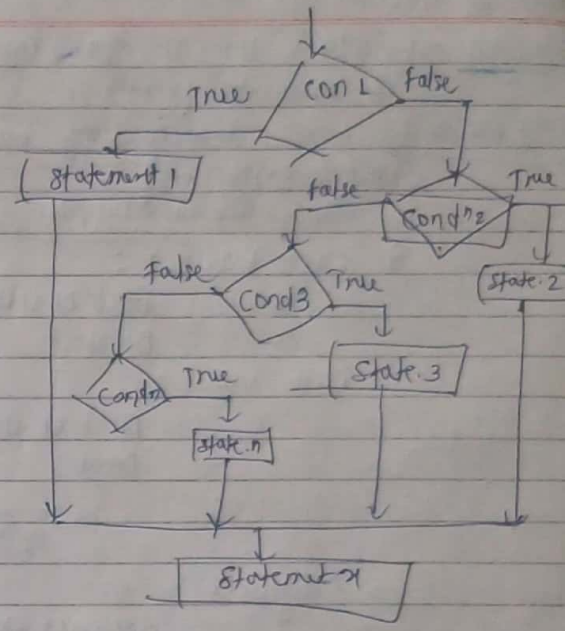
end of program

if (age > 50 && salary < 60k)

(30)

* else - if ladder

```
{
  if (cond 1);
  { statement 1 ;
  }
  else if (cond 2);
    { statement 2;
    }
  else if (cond 3);
    { statement 3;
    }
  else if (cond^n n);
    { statement n;
    }
  else { default statement;
  }
  statement x;
}
```



O/P

if	o/p
A–Z a–z 0–9	Char NO white

marks = 80

Grade is A

(45)
'D' grade

end of program

```
Void main()
{ int marks;
  printf("Enter marks");
  scanf("%d", &marks);
  45 = 0
  if(marks > 80)
  (80)  80 > 80 = 0
  0  printf("Grade is A");
  else if (marks > 70)
  45 = 0  80 > 70  printf("Grade is B");
  else if (marks > 60)
  45 = 0  printf("Grade is C");
  else
    printf("Grade is D");
  printf("End of program");
  getch();
}
```

Not compulsory

Scanned with CamScanner

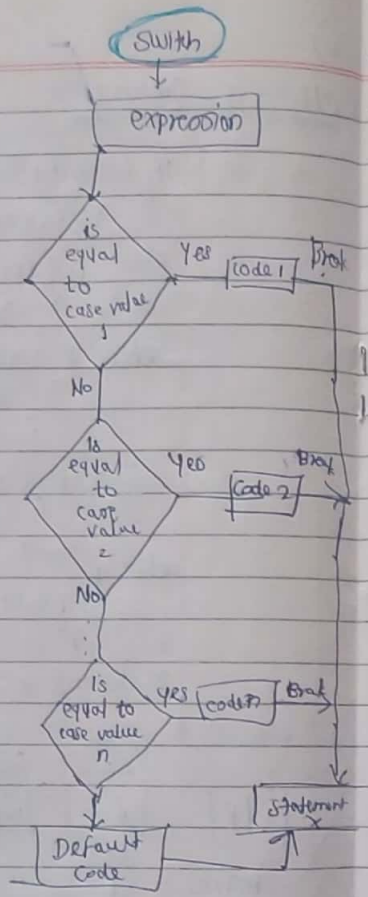
(31) [multiways desicion in c]

*Switch → (Replace long if-else program & execute faster)

only int, char

↳ checking equality

```
switch ( expression )
        ↑value → int / char.
{ case value 1 :
        Block of state-1 ;
        Break ;
  case value 2 :
        Block of state-2 ;
        Break ;
        ↑ (go-out of switch block)
  :
  default
        default state. ;
}
statement x ;
```



#

```
char ch ;
printf("Enter ch");
scanf("%c", &ch);
switch (ch)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        printf("vowel");
        Break ;
    default printf("Not vowel");
}
```

>80 - A
>70 - B
>60 = C
etc

(32)

* Simple calculator using switch.

```
switch (operator)
{
    case '+' :
    case '-' :
    case '*' :
    case '/' :
```

Ex o/p ⇒
Enter operator = +
Enter 2 ~~nos~~ variables = 2, 3
addition is = 5

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

void main()
{ int a, b, sum, sub, div, mul;
  char operator ;

  printf("Enter operator");
  scanf("%c", &operator);

  printf("Enter 2 operands");
  scanf("%d%d", &a, &b);

  switch (operator)
  {
    case '+' : sum = a+b ;
               printf("sum = %d", sum);
               break ;
    case '-' : sub = a-b ;
               printf("sub = %d", sub);
               break ;
    case '*' : mul = a*b ;
               printf("mul = %d", mul);
               break ;
    case '/' : div = a/b ;
               printf("div = %d", div);
```

out of switch

o/p | memory

a	b
2	3

sum	sub
5	

mul	div

operator
+

o/p
Enter operator = +
Enter 2 operand = 2, 3
sum = 5

Scanned with CamScanner

```
        break;
        default:
            printf("Enter valid operator");
    }
    getch();
}
```

(33)

* Loops in c (Repeate of statements many times)

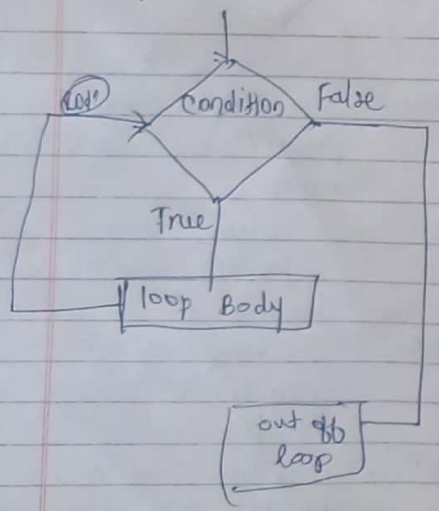
to be used.

① initialization (start) ② cond? (end) ③ out of loop (update expression)

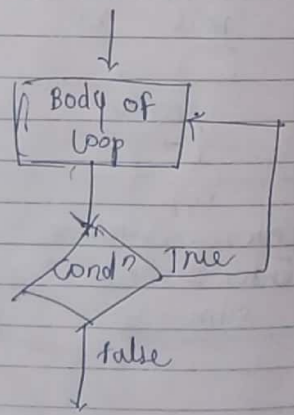
① Entry controlled loop ⇒ (if cond^n is satisfy then control go into loop)

② Exist controlled loop ⇒ (~~1 loop~~

Entry controlled



Exist controlled



(34)

* for loop in c (Entry controlled loop)

```
for(Exp1 ; Exp2 ; Exp3)
    {
    //(Loop Body)
    }
```

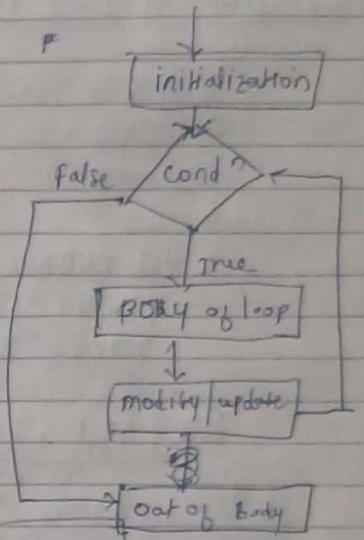
Exp1 → initialization; Exp2 → cond^n.; Exp3 → update/modify

Holo

memory: i

```
cont. false
① ② check ③ update
for(i=1; i<=10; i++)
    {
    ③ printf("i=%d", i);
    }
printf("SH");
```

O/P: i=1 3, i=2 3



i<10 false

i=10 = 10

↳ SH

```
for(i=10; i<10; i++)
    {
    printf("%d", i);
    }
```

```
for( ; ; )
    {
    Hello
    }
```

⇒ ∞ loop

(35)

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

int i=1 -or- for(; i<5; i++)

```
for(i=1, j=0; i<=5; i++)
    {
    printf("%d\n", i);
    }
```

more cond^n; multiple updates i++, j=0

i	j
1	0
2	0
3	0
4	0
5	0

simple cond^n

```
for(i=1, i<=5, j<3; i++)
```

termination cond^n

∞

```
for(i=1, j=0, k=3;
    i<=5, j<=6, k>1;
    i++, j++, k--);
printf("%d %d %d", i, j, k);
```

(i<100 || j<3) ⇒

1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7

(36)

* C Program to print table of a number.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int number, i, a;
    printf("Enter a number");
    scanf("%d", &number);
    for (i=1; i<=10; i++)
    {
        a = number * i;
        printf("%d\n", a);
    }
    getch();
}
```

O/P (for (1-10))

NO same → 2*1	=2
2*2	=4
2*3	=6
2*4	=8
2*5	=10
2*6	=12
2*7	=14
2*8	=16
2*9	=18
2*10	=20

start, termination cond^n, end, update, num, i=11, false, 2*1, 2*2, printf("%d*%d = %d", num, i, a);

memory: number 2, i 2, a 4

O/P: Enter a number 2 → 2 4 6 8 10 12 14 16 18 20

print("%d * %d = %d", num, i, a);
2*1=2

(37) * While loop in C (entry controlled loop)

while (cond^n) — check; true → termination; false; { Body of loop }

initialization; while (cond^n) { statement; update/modify; }

initialize → cond (check) → true → Block of code → update/modify; false

char ⇒ -128 to 127
Range

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int i=1;
    while (i<=10)
    {
        printf("%d\n", i);
        i++;
    }
    getch();
}
```

memory: i 11

O/P: 1 2 3 4 5 6 7 8 9 10

init i → 1; memory; termination; -2*1

int i; while (i<=10) { print("%d", i); i++; } (H.W)

many ways in

(38) * while loop

```
void main()
{
    int i=1;
    while(1)
    {
        printf("%d\n", i);
        i++;
    }
    getch
}
```

O/P: 32767 — 32767 only

while(1) — always true, ∞ loop

while (i==1)

while (i) — 32767, -32767; i=0 → 0 → stop

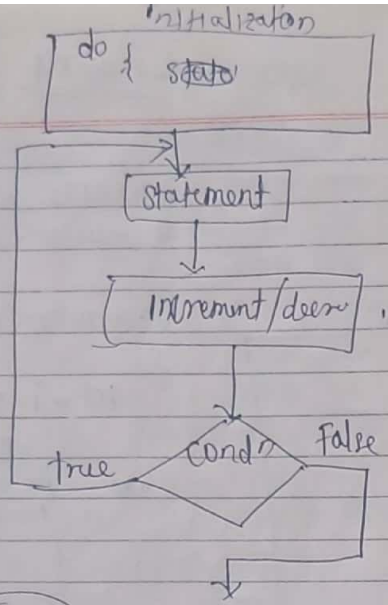
while (i<127 && i==1)
(H.W)

(39).

* Do-while loop in C

[Exist controlled loop]

```
int
do{
    statement
    update/modify
}while (cond^n);
```




```
void main()
{
    int i=0;
    do{
        printf("Hello");
    }
    while(i>0);
    printf("out of loop");
}
```

i [0]

Hello
out of loop

check false

0>0

Exist control loop

when execute statement at least with check cond^n then used

```
void loop()
{
    int i=0
    while(i>0)
    {
        printf("Hello");
    }
    printf("out of loop");
}
```

i [0]

o/p out of loop

false

0>0 = X

Entry control loop

first check condn then execute

(40)

* Diff. betn For, while, Do while

	For	while	Do while
-	Entry control loop	Entry control loop	Exist control loop
-	for(cond^n, cond^n, update) { Body loop }	while (cond^n) { Body loop }	do { statement; } while (cond^n);
-	Body of loop never existing while condn is false	—||—	Do statement at least one
-	Repeat the statement exactly known (iteration) ex. (1—10)	while(1) not known iteration (end)	—||—

Scanned with CamScanner

(41)

* break statement in c

→ loops & switch statement

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

void main()
{
    int i, a, sum=0;

    for (i=1 ; i<=5 ; i++)
    {
        printf("Enter a num ");
        scanf("%d", &a);
        if (a<0)
            { break ;
        }
        sum = sum + a ;
    }
    printf("sum = %d", sum);
}
```

memory

a	i	sum
1	1	1
5	2	6
	5	
	6	

o/p

Enter a num 1

Enter a num 5

Enter a num −10

6

```
void main()
{
    int i, a, sum = 0 ;
    while (1)
    {
```

not known value

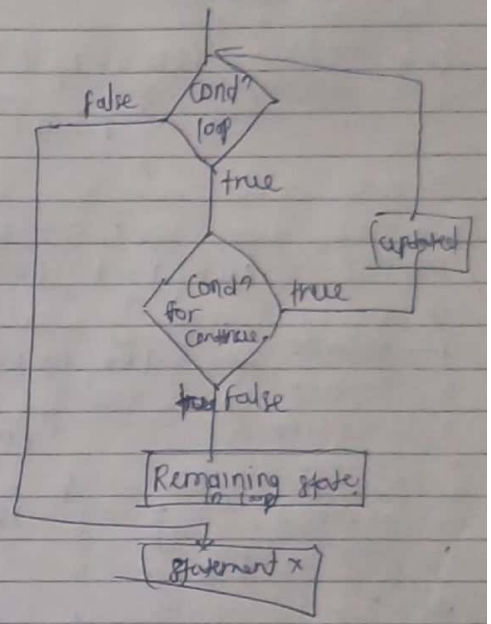
while + do-while

a	sum	o/p
1	3	1
2+1	4	2

(42)

* Continue statement in c – (skip the statements)

```
for ( __ ; __ ; __ )
{
    statement 1 ;
    statement 2 ;
    if (
        continue ;
    }
}
```




```
void main()
{
    int i, a, sum=0;

    for( i=1 ; i<=5 ; i++)
    { printf("Enter an integer ");
      scanf("%d", &a);
      if ( a <0)
          { continue ;
          }
      sum = sum + a ;
    }
    printf("sum= %d", sum);
}
```

o/p

Enter an integer

=1

2

=−10

−2

=0

[Sum = 3]

i	a	sum
1	1	1
2	2	3
3	4	3
	−10	
4	−2	9
5	0	9
6		

Scanned with CamScanner

(43)

* Nested loops in C -

Repeat something with loop

Loop inside loop

for & while

```
for(i=1; i<=5; i++)
{
    printf("*");
}
```

= (O/P)

```
* * * * *
```

(Loop)

```
for(j=1; j<=6; j++)
{
    for(i=1; i<=5; i++)
    {
        printf("*");
    }
    printf("\n");
}
```

(1) (2) (3) (4) (5) (6)

memory

j	i
1	1 2 3 4 5 6
2	1 2 3 4 5 0
3	1 2 3 4 5 0 7
4	1 2 3 4 5 6
5	1 2 3 4 5 0
6	1 2 3 4 5 0

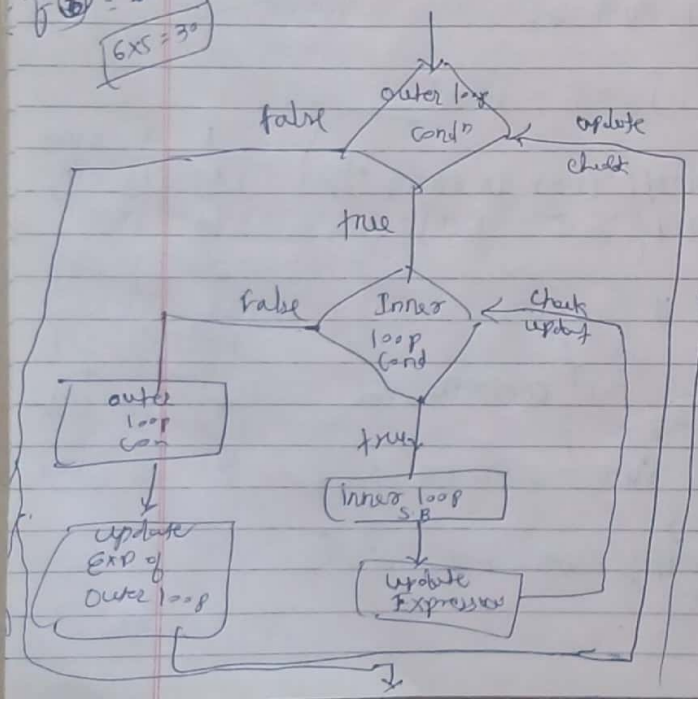
$\boxed{i=6}$

$\boxed{6 \times 5 = 30}$

```
*****
*****
*****
*****
*****
*****
```

for (6)

```
for (Init, cond, update/action)
{
    for (I; C; ↑/↓)
    {
        inner loop
    }
    outer loop.
}
```



(44)

* Nested while

```
while (cond^n)
{
    while (cond^n)
    {
        Inner loop { inner stm
    }
    outer loop;
}
```

(45)

* Nested do-while loop in C

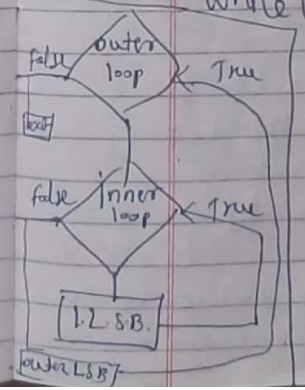
```
do
{
    do
    {
        } while (cond^n);
    } while (cond^n);
```

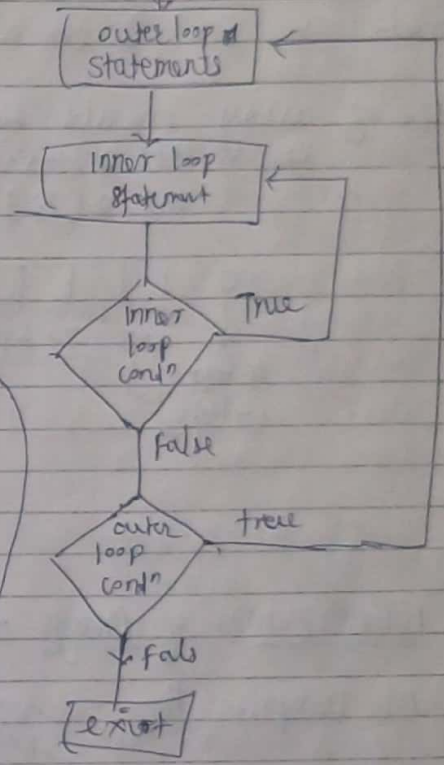
outer loop

inner loop

```
int i=1, j=3;
do
{
    do
    {
        printf("%d", j);
        j--;
    }
    while (j>0);
    i++;
    printf("%d", i);
}
while (i<4);
```

(1)





i	j
1	3
2	2
3	1
4	0
	-1
	-2

memory

addr ⇒ hexadecimal

Need? ⇒

(46) [~~collections of homogenious data items~~

* Arrays in C

datatype	name of array	[size of array]
①	②	③

cond^n ⇒ ① store one data item but all of data type is same type.

ex- int a[5];
↳ a(1,2,3,4,5)

↳ is a collection of homogenious data items which are stored in a contiguous ~~data~~ memory location

- size of array should be a +ve expression
  ↳ int a[-5] ✗ int a[2*2] ✓
  int a[5] ✓ int a[2/2] ✓

a [| | | |]
100 101 102 103 104

↓ 4×5 = 20 byte

~~4 size~~ (4 byte)

USING MACROS

int a[N] ✗
↓ used
macros
# define N 5;
main()
{
int a[N]; ✓
}

(47)

* initialization of Arrays -

(i) At compile time - char/float a[5] int a[5] — declaration

① int a[5] = {0, -1, 11, 10, 2}
② int a[] = {0, 1, 2, 0, -1, 6, 7}
automated calculation
③ int a[5] = {0, 1, -1}
4 5 → set '0'
④ int a[5]; → any garbage value
int a[5] = {0, 1, 2, 3, 4, 5, 6, 7} ✗ error

memory
a [| | | |]
-4 -5 -1 -2 -3

(ii) At Run time - large data

int i;
int a[5];
printf("Enter the element of array");
O/P

for (i=0; i<5; i++)
① ② ④
{
scanf("%d", &a[i]);
}

i [ ]
100

a [1 | 2 | 3 | 4 | 5]
2000 2004 2008 2012 2016

int i;
int a[100];
for (i=0; i<100; i++)
{
if (i<30) → first 30 = 1
a[i] = 1;
else → all = 0
a[i] = 0;
}

(48)

* memory Representation & Accessing of array -

[memory address 2000 2001 — 2013]
(data stored in continous)

internal pointer
a (array)
2000
Base address

↓ a[index] = a[0] = 1 ✓ first elem

int a[5] = {1, 10, 0, -1, 3};
↓ ↓
4 byte * 5 byte = 20 bytes

start → 0 1 2 3 4 → index
a → [1 | 10 | 0 | -1 | 3]

1 10 0 -1 3

2000 2004 2008 2012 2016 2020

Base add.

a[3] = -1 → forth index
index
Accessing

→ 1 D array

- access any element of array in constant time.

accessing → Base add + index * size of.

2000 + 3 * 4 = 2000 + 12 = 2014

(49)

* - Array is a collection of more than one data item of same data.

- All the data item are stored in contiguous memory location.

- Numbers of data items array hold is size of array.

- Once size has declared, it can't be changed at run time (Fixed size).

- Index starts from 0.

- Known as derived data type

- Accessing of any element is faster using Index of array

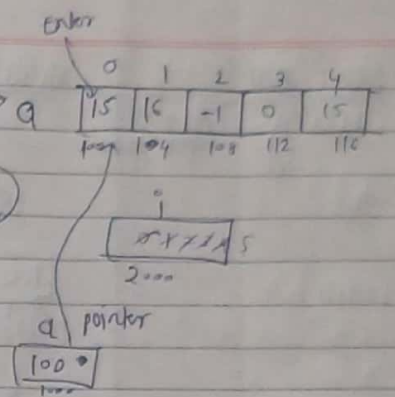
- It allows to store data in multidimensional form

- Inserting & deleting elements from array is costly

- No bound checking in C. (any array value

```
void main ()
{ int i;
  int a[5];
  for(i=0; i<5; i++)
  {
    scanf("%d", &a[i]);
  }
  for(i=0; i<5; i++)
  {
    printf("%d\n", a[i]);
  }
}
```

≡ Print cmd for user



B + i * size of

0 + 0 * 4

O/P
15
16
-1
0
15

— all same —

```
for(i=4; i>=0; i--)
{
  printf("%d\n", a[i]);
}
```

i
4

a

a(4) = 100 + 4 * 4
= 116 112

O/P
15
0
-1
16
15

Reversed order

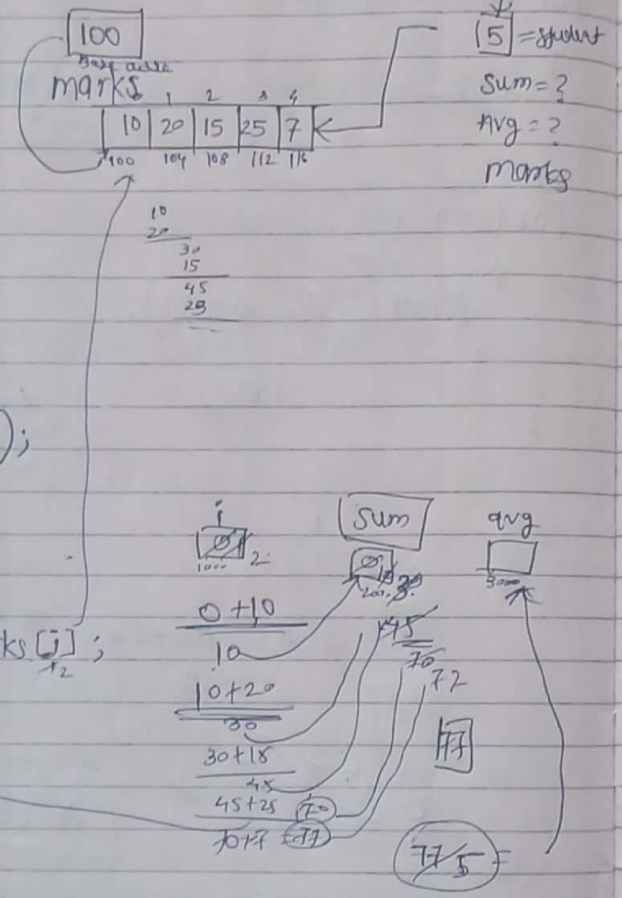
(50)

* WAP to Read marks of 5 students. Calculate sum & average using Arrays.

(Code)

```
#include <stdio.h>

void main()
{
  int sum=0, avg, i;
  int marks[5];

  for (i=0; i<5; i++)
  {
    scanf("%d", &marks[i]);
  }

  for(i=0; i<5; i++)
  {
    sum = sum + marks[i];
  }

  avg = sum/5;

  printf("%d", avg);
}
```

(memory)

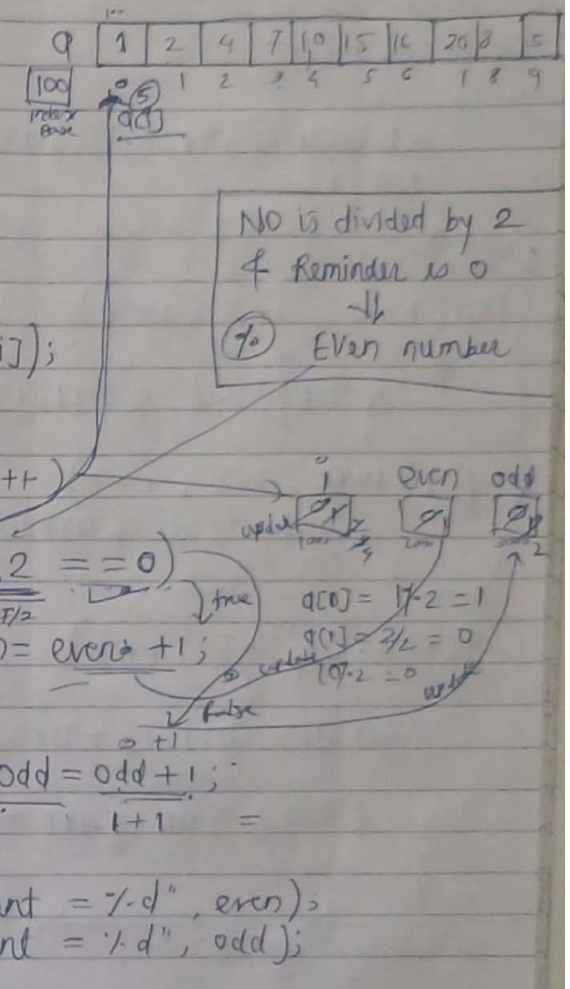


(O/P)

(51)

* WAP to Read an array of 10 integers & count total no. of even & odd elements

```
void main()
{
  int q[10], i;
  int even=0, odd=0;

  // Read values
  for (i=0; i<10; i++)
  {
    scanf("%d", &q[i]);
  }

  // accessing
  for(i=0; i<10; i++)
  {
    if (q[i] % 2 == 0)
    {
      even = even + 1;
    }
    else
    {
      odd = odd + 1;
    }
  }

  printf("Even element = %d", even);
  printf("Even element = %d", odd);
```



1-Row multiple Coloum
1D arrays

(52)

* WAP to Read two array of size 5 & store sum of these arrays into third array

	100	104	108	112	116
a (100)	10	0	-1	1	2
a2 (104)	0	5	4	3	1
a3 (100)	10	5	3	4	3

i (0)

```
void main()
{
  int arr1[5], arr2[5], arr3[5];
  int i;

  for(i=0; i<5; i++)
    scanf("%d", &arr1[i]);

  for(i=0; i<5; i++)
    scanf("%d", &arr2[i]);

  for(i=0; i<5; i++)
    arr3[i] = arr1[i] + arr2[i];

    printf("sum of arr = %d is %d\n", i, arr3[i]);
}
```

accessing ⇒

a[0][2] ⇒ (0,2)

printf("%d", a[0][2]);

Need? ⇒ multiple array operation

(53)

* Two Dimentional Arrays

Datatype name of array [Row] [Coloum]

int marks [3] [5] ;

	C0	C1	C2 (student)	C3 (sub)	C4
R0	0,0	0,1	0,2	0,3	0,4
R1	1,0	1,1	1,2	1,3	1,4
R2	2,0	2,1	2,2	2,3	2,4

matrix 3*5 — 15 elem — *4 — 60 bytes

marks [1] [2]

(54) * initialization

① At compile time — int a [ ] [ ] =

int a[2] [3] = {0,0,0,1,1,1};

2*3 = 6 elements → 4 (int) → 24 byte

	0	1	2
Row 0	0	0	0
1	1	1	1

or

{{0,0,0},{1,1,1}}

② At Run time —

int a[2][3];

for(i=0; i<2; i++)
{
  for(j=0; j<3; j++)
  {
    scanf("%d", &a[i][j]);
  }
}

	C0	1	2
Row 0	0,0	0,1	0,2
1	1,0	1,1	1,2

(55)

* memory Representation & accessing of elmt

int a[2][3] = {{1,2,0},{-1,5,4}}

	0	1	2
Row 0	1	2	0
Row 1	-1	5	4

Row0 Row1

			1	2	0	-1	5	4							
			100	104	108	112	116	120							

Scanned with CamScanner

(56)

* To print 2 Array (matrix) & Calculate sum.

	0	1	2
0	10	-1	3
1	0	2	5

```
int a[2][3], i, j;
int sum=0;

for (i=0; i<2; i++)
{
    for(j=0; j<3; j++)
    {
        scanf("%d", &a[i][j]);
    }
}

for(i=0; i<2; i++)
{
    for(j=0; j<3; j++)
    {
        printf("%d\t", a[i][j]);

        sum= sum+ a[i][j];
    }
    printf("\n");
}
printf("sum%d", sum);
}
```

i j sum

0 + 0

10	-1	3	0	2	5
Row 0			Row 1		

\t = tab space

sum = 0+0 (10) =

printf("\n") ⟹ // space

0 t -1 3
2 5

(57)

* WAP to print Transpose of matrix

↳ changing Row = Coloum & coloum into Row

(1st method)

```
int a[2][3], i;

//Read matrix
for(i=0; i<2; i++)
{
    for(j=0; j<3; j++)
    {
        scanf("%d", &a[i][j]);
    }
}

//Transpose
for (i=0; i<3; i++)
{
    for(j=0; j<2; j++)
    {
        printf("%d\t", a[j][i]);
    }
    printf("\n");
}
```

↳ interchange Row ⇄ coloum

	0	1	2
0	1	2	3
1	4	5	6

2x3

⇓ Transpose

	0	1
0	1	4
1	2	5
2	3	6

3x2

i j

00 = 1
10 = 4
01 = 2

Now B = [1 4; 2 5; 3 6]

o/p

1	4
2	5
3	6

Scanned with CamScanner

*(58)

* WAP sum of individual Rows & columns.

```
int a[3][3], i, j, SR, SC;
```

SumRow, sumcolumns

```
for(i=0; i<3; i++)
{
  for(j=0; j<3; j++)
  {
    scanf("%d", &a[i][j]);
  }
}
```

scan

```
for (i=0, i<3; i++)
{
    SR = SC = 0   // initialize
    for(j=0; j<3; j++)
    {
        SR = SR + a[i][j];
        SC = SC + a[j][i];
    }
    printf("SR=%d, SC=%d", SR, SC);
}
```

SR = 4 SC = 3
SR = 20 SC = 10
SR = 5 SC = 14

```
printf("transpose of matrix");
for(i=0; i<3; i++)
  (j=0; j<3; j++)
    printf("SR=%d SC=%d", SR, SC);
    printf("\n");
}
```

SR	SC
4	0
1	0
6	15
5	14

(59)

* point addition of matrix

```
int a[2][3], b[2][3],
    c[2][3], i, j;

for(i=0; i<2; j++)
{
  for(j=0; j<3; j++)
  {
    scanf("%d", &a[i][j]);
  }
}
~~for for~~
for(i=0; i<2; i++)
{
  for(j=0; j<3; j++)
  {
    scanf("%d", b[i][j]);
  }
}

for (i=0; i<2; i++)
{
  for(j=0; j<3; j++)
  {
    c[i][j] = a[i][j] + b[i][j];
    printf("%d\t", c[i][j]);
  }
}
```

(60)

$a_{(3\times4)} * b_{(3\times2)}$ ✗ $a_{2\times3} * b_{3\times2}$

* Print multiplication of 2 matrix

$$c = \begin{bmatrix} (1\times1)+(5\times0)+(6\times1) & 2+5\times0 & 3+25+12 \\ 7 & 7 & 40 \\ (2\times1)+(2\times0)+(5\times1) & (2\times2)+(2\times0)+(1\times2) & (2\times3)+(2\times5)+(5\times2) \\ 7 & 6 & 26 \\ (1\times1)+(7\times0)+(5\times1) & (1\times2)+(7\times1)+(5\times0) & (1\times3)+(7\times5)+(5\times2) \\ 6 & 9 & 48 \end{bmatrix}$$

$$a = \begin{bmatrix} 1 & 5 & 6 \\ 2 & 2 & 5 \\ 1 & 7 & 5 \end{bmatrix}_{3\times3} * b = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 5 \\ 1 & 0 & 2 \end{bmatrix}_{3\times3}$$

$$\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}_{3\times3}$$

Rules

- Columns of first matrix = Rows of 2nd matrix

$A_{m\times n} * B_{p\times q}$ if $(n == p)$ and

- 1st matrix colums multiplebcy Row of matrix 2

(61)

$$a = \begin{bmatrix} 0 & 5 & 2 \\ -1 & 1 & 0 \\ 7 & 4 & 3 \end{bmatrix}_{3\times3} \quad b = \begin{bmatrix} 1 & 2 \\ 0 & 4 \\ 3 & 5 \end{bmatrix}_{3\times2}$$

$$c = \begin{bmatrix} 6 & 30 \\ -1 & 2 \\ 16 & 45 \end{bmatrix}_{3\times2}$$

```
for(i=0; i<3; i++)
{
  for(j=0; j<2; j++)
  {
    sum=0;
    for(k=0; k<3; k++)
    sum = sum + a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

i j k sum

$$\begin{bmatrix} (0\times1)+(5\times0)+(2\times3) & (0\times2)+(5\times4)+(2\times5) \\ 6 & 30 \\ (-1\times1)+(1\times0)+(0\times3) & (-1\times2)+(1\times4)+0 \\ -1 & -2+4 \\ (7\times1)+(4\times0)+(3\times3) & (7\times2)+(4\times4)+(3\times5) \\ 7+0+9=16 & 14+16+15=45 \end{bmatrix}$$

i j k sum

$0<3$ $0<2$ $0<3$ $1<3$

sum = sum + a[i][k] * b[k][j];

$0 + a[0][0] * b[0][0]$

start of matrix

$0 + 0 * 6 = 0$ sum

$c[i][j] = sum$

$c[0][0] =$

$(0\times1)+(5\times0)+(2\times3)$

$(5*1)+(2*0)+3$

$5+0+3$

$$\begin{bmatrix} i,k_0 & i,k_1 & i,k_2 \\ i,k_0 & i,k_1 & i,k_2 \\ i,k_0 & i,k_1 & i,k_2 \end{bmatrix} * \begin{bmatrix} j,k_0 & j,k_1 & j,k_2 \\ j,k_0 & j,k_1 & j,k_2 \\ j,k_0 & j,k_1 & j,k_2 \end{bmatrix}_{3*2}$$

$$= \begin{bmatrix} i,j_0 & i,j_1 \\ i,j_0 & i,j_1 \\ i,j_0 & i,j_1 \end{bmatrix}_{3*2}$$

Scanned with CamScanner

(62)

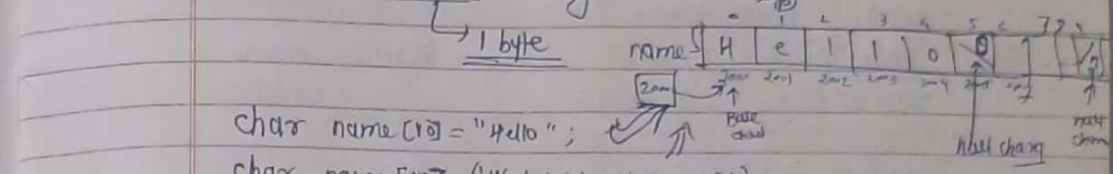
* Strings in C - [array of characters] ("___")

(T.S)

data type array name [size]

char string name [size];

1 byte



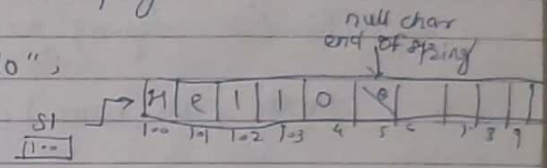
char name[10] = "Hello";

char name[10] = ('H','e','l','l','o','\0');

(63)

~~printf~~ Read of Run (scanf, gets) in strings

char s1[10] = "Hello";




```
char s1[10];
printf("Enter string");
scanf("%s", s1);
printf("%s", name);
```

(scan) scanf("%s", s1); → [can not consider space] → drawback → used gets fn

(end) printf("%s", name);

gets()

```
char s1[10];
printf("Enter string");
gets(~~no~~ s1);
printf("%s", name);
```

char s1[10]; → [Buffer size (11111 \0)]

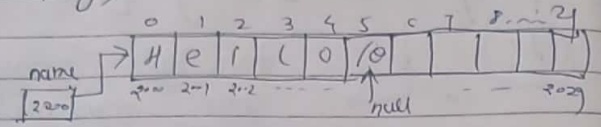
gets(~~no~~ s1); → space include → end

drawback → [Buffer overflow] → overwrite

(64)

* printf & puts (print a string)

char name[30];




```
printf("enter name");
scanf("%s", name);
printf("%s", name);
```

⟹ Enter name =
Hello.

printf("%2S", name); ⇒ He.

printf("%-2S", name); ⇒ _ _ H e.
blank

* puts (pre defined fun)

~~putt~~ puts(name); ⟹ Next line prints op.

⟹ He _ _ He
Hello.

(65)

* C program to read & print a string.

```
#include <stdio.h>

void main()
{
    char name[30] = "Hello";
    printf("%s", name);
```

⟹ Hello

-OR-

```
    printf("Enter name");
    scanf("%s", name);
    printf("%", name);
```

⟹ enter name
Hello
⟹ Hello

(66)

* C Program to find length of a string

string.h
↳ predefined fun

use strlen = length of string

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

void main()
{ char name[30];
  int count = 0;
  printf("Enter name");

  gets(name);
  count = strlen(name)
  printf("%d = string length", count);
}
```

(1) 0 1 2 3 4 5
H e L L o \0
1000 1001 1002 1003 1004 1005 ...

strlen(char* str)
(unsigned int)

with strlen fun.

OR

without strlen fun

```
while (name[i] != '\0')
{
    count++;
    i++;
}
printf("%d", count);
}
```

count: 0 → 1 → 2 → 3 → 4 → 5
i: 0 → 1 → 2 → 3 → 4

Scanned with CamScanner

— Buffer over flow.

(67) strcat → combine

* program to concatenate two strings

(destination) → S1 | H | e | y | \0 | | |

strcat (destination, source)
S1 S2

(source) → S2 | H | e | l | l | o | \0 |

~~int~~
char S1[30] = "Hey";
char S2[7] = "Hello";

with strcat fun

strcat (S1 , S2);
destination source

printf(" %s ", S1);
destination

= "Hey Hello"

H	e	y	H	e	l	l	o
0	1	2	3	4	5	6	7

strcat (S1,S2,2)
⇒ Hey He

int len1, len2; (Init)
~~length~~ len1 = strlen (S1);
len2 = strlen (S2);

len1	len2	i
3	5 (6)	0

without strcat fun

for (i=0, i<=len2 ; i++)
0 <= len2
{
S1[len1+i] = S2[i]
}

printf(" %s ", S1); → Hey Hello.

(68) strcmp
strncmp

* Compare 2 strings
↳ [chara. by chara]

str1 = computer
str2 = computer

int strcmp (const char * str1, const char * str2)
(Left string) (Right string)

↳ 0 ⇒ same
+ve ⇒ when first char that does not match is greater in string 1
-ve ⇒ when the first char that does not match is less in string

with
int value;
char S1[ ] = "Hello";
char S2[ ] = "Welcome";

value = strcmp (S1,S2);

if (value == 0)
same
else
not same

not same

printf (" %d", value);

without

~~char S1[7]~~ → | H | e | l | l | o |

S2[i-1] | W | e | l | c | o | m | e |

int flag = 0;
for(i=0; S1[i] != '\0'
|| S2[i] != '\0'; i++)
{
if(S1[i] != S2[i])
{
flag = 1;
break;
}
}
if (flag == 1)
{ printf("not same");
}
else { same; }
}

(69)

* program to reverse a string → strrev

(swapping)

S1

s	h	u	b	h	a	m	\0

(with)

```
char S1[20] = "shubham";
strrev (S1);//char * strrev (char * str)
```

⇓

m	a	h	b	u	h	s	\0

```
printf("%s", S1);
```

(without)

```
int l, i;
char ch;
char S1[30] = "shubham";
l = strlen (S1);
for (i=0; i<l/2; i++)
{
    ch = S1[i];
    S1[i] = S1[l-1-i];
    S1[l-1-i] = ch;
}
printf("%s", S1);
```

(7/2) = 3.5 → 3

swapping

S1[0]	S1[6]
S1[1]	S1[5]
S1[2]	S1[4]
~~S1[3]~~ ✗	

S1[i] S1[l-1-i]

#

ch	l	i
~~s~~	~~7~~	~~0~~
~~h~~ u	6	~~1~~ 2

(70)

* convert string from upper case to lower case

↗ not standard fu

* strlwr ( ) lower case
* strlupr ( ) upper case

shubham
SHUBHAM
modify string

```
char * strlwr ( char * str)
```

(with)

```
char S1[30] = "ShuBham";
strlwr [S1];
printf("%s", S1);
```

ShuBham
⇓
shubham

S	h	u	B	h	a	m

S1

(without)

```
for(i=0; S1[i]!='\0'; i++)
{
    if ( S1[i] >= 65 && S1[i] <=90)
    {
        S[i]= S[i] + 32
    }
}
printf("%s", S1);
```

'A' 'Z'

ASCII

A = 65 (+32) a = 97
B = 66 (+32) b = 98
⋮
Z = 90 (+32) z = 122

(add)

Scanned with CamScanner

*

(71) [indirection / operation / re]

(Hexadecimal = %x)

* Pointers in C - [derived data types]

- special variables which contains of address of any variables.

a ← Name
[10] ← value
1000 ← address

Float x, y;
int a, *ptr;
ptr = &x;
ptr = &a;

points to

→ Data type * pointer name;

Initialization
- int * p;
// float *p;

P
[1000]
2000

- p = &a ; // (p ptr points to a.)

int *p = &a ;
int a=10 , *p = &a ;

(72) (Reference operator)

* & (address of) and * (Indirection) operator

↗ *point at value

# int c;
int a=10, b=9;
int *p, *q;
p = &a;
q = &b;
# c = *q;
printf(" value of a = %d ", a);
printf(" a=%d ", *p);

a [10] 1000
b [9] 2000
p [1000] 200
q [2000] 100

= a = 10
⇒ [10]

printf(" a=%d ", &p) ; ⇒ 200
printf(" a=%x ", p) = 1000
printf(" a=%x ", &a); = 1000

printf(" c = %d ", c); = ? (9)

(73) ↗ 2/4 byte

* pointer Assignment -

int a = 10, b = 11;
int *p, *q;
(static)
p = &a; q = &b;
# q = p;
∴ *q = *p ; // error
printf(" q = %d, %d %d ", a, *p, *q);

a [10] 1000
b [11] 1046
p [1000] 2000
q [1046] 3000

(74) // (store add. of another pointer variable)

* pointer to pointer (double pointer) :-

int a = 10;
int *p = &a ;
int **q = &p ;

printf(" a = %d %d %d ", a, *p, **q)
↓
((&p))
((250)) = 1000 = 10

a [10] 1000 int
p [1000] 850 int *
(pointer to variable)
q [850] 1050 int **
(pointer to pointer)
[1050] 2040
***

(75) * pointer Arithmetic (Addition) ⇒ pointer base add + 1 * data type byte.

int a = 10;
int *p = &a ;

P = p+1;

a [10] 1000
p [1000] 1046 (1004)
4 byte int data type
1000 1001 1002 1003 1004 1005 1006 1007 1008
New

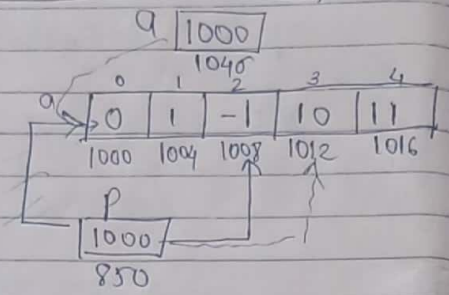
∴ p+1 * datatype = 1000 + (1 * 4) = [1004]

Scanned with CamScanner

printf("%d", *p) ; = 1004

$$\boxed{p+n = p + n * \text{Size of data type}}$$

int a[5] = {0, 1, -1, 10, 11}
int *p = &a[0];



printf("%d", *p); = 0

p = p+2 ; = -1, 1008
= 1000 + (2*4)
p = 1008

printf("%d", *p); -1

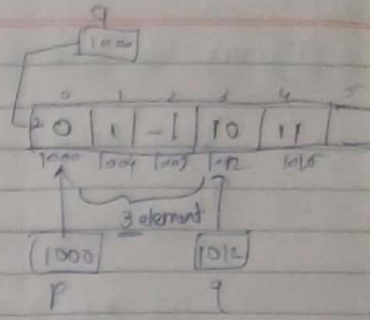
p = &a[0];
p+n = &a[0+n];

p = p+1;
printf("%d", *p) ; ⇒ 10, 1012

(76)
* Subtraction of pointer arithmetic.

int a[] = {0, 1, -1, 10, 11} ← constant pointer
int *p = &a[0];

int *q = &a[3];



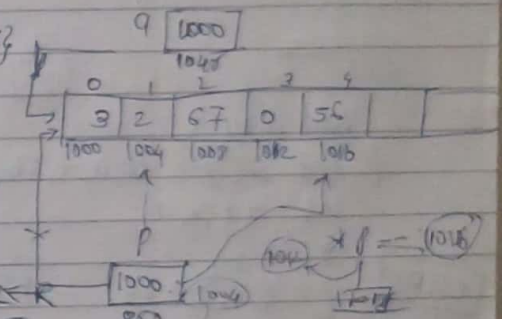
d = q - p ;
↳ 1012 - 1000 ⇒ 12/4 ⇒ 3 (int)
(Size of datatype)

q = q - 2 ; (address)
↳ 1012 - 2*4 = 1004

d = p-q
*q = 25
d = q-p;
*p = 27;
q = q-3;
p = p+3
d = p-q

(77)
* pointer arithmetic (Increment & Decrement)

int a[] = {3, 2, 67, 0, 56}
int *p ;
p = a; / (p = &a[0])



p++ ; / (p = p+1;)
1000 + (1*4) = 10

(postfix)
printf("%d", *p); = 2

*Ptr → Return the value then increment
*++p → Inner value & this value return

(*p--) ⇒ 3
(995)
garbage value

printf("%d %d", *p++, *p+1);
1004 2 1008 67 3

("%d %d %d", *p--p, *--p, *--p);
56 1016 10 1012 67 1008

(78)

* program ⇒ pointer.

```
void main()
{
    int a[] = {10,11,-1,56,67,5,4};

    int *p, *q;
    p = a;
    printf("%d\n", *p);
    printf("%d %d %d\n", (*p)++, *p++, *++p)
    q = p+3;
    printf("%d\n", *q-3);
    printf("%d\n", *--p+5);
    printf("%d\n", *p+*q);
}
```

→ indirection operator

q = 100; a: 0 1 2 3 4 5 6 = 10 11 -1 56 67 5 4; 100 104 108 112 116 120 124

p = 100; q = 120

(*p)++ → 10; *p++ → 11; *++p → 5; *p = 11

q = p+3 ⇒ 5 (120); 11+1; 5-3 = 2 (120)

*--p+5 ⇒ 16; 11+5 = 16

11+5 = 16

```
void main()
{ int a[] = {10,11,-1,56,67,5,4};
  int *p*q;
  p = a;
  q = &a[0]+3;
  printf("%d %d %d", (*p)++, (*p)++, *(++p));
  printf("%d", *p);
  printf("%d", (*p)++);
  printf("%d", (*p)++);
  q--;
  printf("%d", (*(q+2))--);
  printf("%d", *(p+2) - 2);
  printf("%d", *(p+2) - 2) - 1);
```

q = 100; a: 0 1 2 3 4 5 6 = 10 11 -1 56 67 5 4; 100 104 108 112 116 120 124

p = 100 (1000); q = 103 (2000)

error

q = &a[0]+3; ⇒ 10+3 = 13

(79)

* void main()

```
{
  char str[] = {"shubham shastri"};
  char *ptr = str;

  printf("%c", *ptr);
  printf("%ch", *(ptr++ +1));
  printf("%c\n", *((ptr-- +5)-1) +3);
  printf("%c\n", *(++ptr +10) - 32);
  printf("%c %c %c", *ptr, *++ptr, *--ptr);
}
```

const ⇒ only Read not modify value

ptr = 100; str = 100; s h u b h a m s h a s t r i

printf("%c", *ptr); ⇒ *(100) ⇒ s

printf("%ch", *(ptr++ +1)); ⇒ *(100+1) = *(101) = h

printf("%c\n", *((ptr-- +5)-1) +3); ⇒ *((101+5)-1)+3) ⇒ d

printf("%c\n", *(++ptr +10) - 32); ⇒ *(101+10) - 32 ⇒ *(S ← 32) = S

printf("%c %c %c", *ptr, *++ptr, *--ptr);

*101 = h; L: S; R: S

(80) void pointer :- it not having any associated data type

* empty

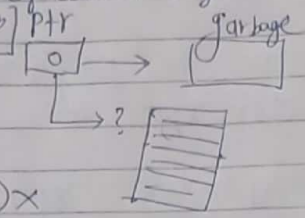
```
int *ip ; int a = 5.
float *fp ; float b
char *cp ; char c

void *vp      vp = &a ;
              vp = &b ;
              vp = &c ;

printf("%d", *(int*)vp); ⇒ 5
printf("%f", *(float*)vp); ⇒ 1.5
printf("%c", *(char*)vp); ⇒ 0
```

a = 5 (100); vp = 100; b = 1.5 (100); c = '0' (500)

(81) NULL

* Null pointer in c [we cannot dereferenced any memory location]

← int *ptr = NULL ;
↳ 0

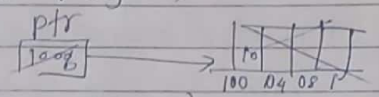


printf("%d", ptr); ⇒ 0.
printf("%d", *ptr); ⇒ crash ×

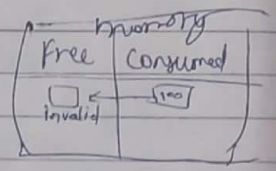
ptr = (int *) malloc (5 * size of (int*));
// at Run time
if (ptr != NULL) { } else { }
// printf("%d", *ptr); ⇐ crash ×

(82) Safe

* Dangling pointer in c (distruction of object)
↳ Hanging



int *ptr = (int *) malloc (size of (int*));
*ptr = 10;
printf("%d", *ptr); = 10.
Free (ptr); ⇒ // deleted memory
dangling pointer.



① ptr = NULL;

if (ptr == NULL);
{ } else { }

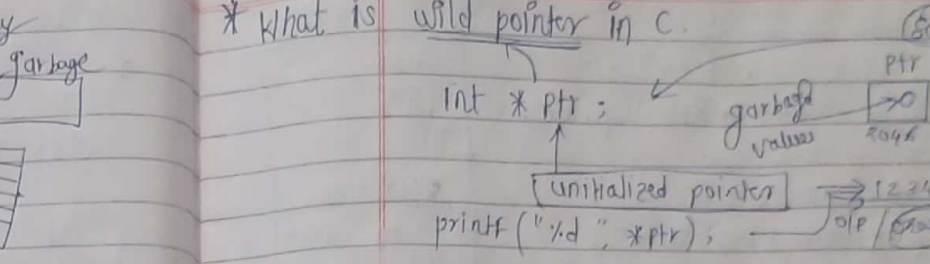
— pointer is pointer to the free memory location is called dangling pointer

②

(83)

* What is wild pointer in c.

int *ptr ;



printf("%d", *ptr);

int *ptr = (int *) malloc (sizeof (int));
*ptr = 5;

printf("%d", *ptr); ⇒ 5

include <stdio.h>
include <stdlib.h> // malloc used

void main ()
{ int *ptr;
printf("%d\n", *ptr);
}
⇒ crash / 0 / wild pointer

void main ()
{ int *ptr = NULL;
printf("%d\n", *ptr); ⇒ 0
}

int a = 9;
int *pre = (int *) malloc (sizeof (int));
*ptr = 67;
printf("%d\n", *ptr); ⇒ 67

Scanned with CamScanner

(84)

* Functions in C - Introduction

user defined fun

```
void sum()        → fun defination
{
    int a, b, sum=0;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    sum = a+b;
    printf("sum = %d\n", sum);
}
```

function

RAM

main
51

void sum		
a	b	sum
2	3	5

o/p = 5.
— 5.
= 5

pre defined fun

```
void main()
{
    sum();      → calling fun
    ---
    sum();      // multiple time addition used only using calling function
    ---
    sum();
}
```

reusability

sum/sub/div/mult
prime/even/odd
more more

```
#include <stdio.h>
void sum();   // fun decl^n
void main()
{   sum();    // calling fun
}
void sum()
{   int a, b, sum=0;
    printf("Enter two number");
    scanf("%d %d", &a, &b);
    sum = a+b;
    printf("sum = %d\n", sum);
}
```

(85)

* Fun declaration

(86)

* Function defination in C

```
void sum();     // decl^n. fun
void main()
{
    sum();      // calling fun
}
void sum()      // def^n fun
{   int a, b, sum=0;
    printf("Enter two numbers");
    scanf("%d %d", &a, &b);
    sum=a+b;
    printf("sum = %d\n", sum);
}
```

data type → (int, int)

int a, int b

fun header

return type frame (argument with data type)

fun body

- statements
- return statement

passing address → &x &y

pointer used (int *x, int *y)

(87)

* Call by values & Call by Reference -

address

```
void fun(int, int);
void main()
{   int x=5, y=7;
    fun(x, y);      // fun calling
    printf("x=%d, y=%d", x, y);
}
void fun(int x, int y)
{   x=7;    // local variables
    y=5;
    printf("x=%d y=%d", x, y);
}
```

parameters

actual para (arguments)

formal para.

call by value

main

x	y
5	7
100	200

call by value

call by reference

fun

x	y
5	7
7	5
100	200

o/p: x=7, y=5

void Return

Scanned with CamScanner

Return type	Fun Name	Arguments

(88)

* classification of Fun.

1. No Argument without Return type
2. No Argument with Return type
3. with Argument without Return type
4. with Argument with Return type.

* (1) No Argument without Return type

```
void  Fun name ( void )
(No Return type)  (No Argu.)
```

```
void Fun(void);   //calling
void fun ()
{
}
```

```
void Fun();
```

```
void sum(void);   // decl^n fun
```

```
void main()          // calling fun
{ sum();
}  ← exist
void sum()           // called fun
{ int a, b, sum = 0;
  printf(" Enter numbers ");
  scanf(" %d %d ", &a, &b);

  sum = a+b;
  printf(" sum = %d ", sum);
}
```

main()

sum: a, b, sum

exist main then access

H.W

sum, sub, mul, div all fun calling main fun.

(89)

* (2) NO argument with Return type

```
int sum(void),   // decl^n.

void main()      //calling
{ int = s;
  s = sum();
  printf(" sum = %d ", s);
}

int sum()        // called
{ int a, b, sum = 0;
  printf(" enter No's ");
  scanf(" %d %d ", &a, &b);
  sum = a+b;
  // printf(" sum = %d ", sum);
  return sum;
  printf(" inside fun^n");  → Not exist
}
```

with Return type

main: s [12]

sum fun: a [5], b [7], sum [12]

vanish after Return

(90) * with argument without Return type

```
void sum(Float, Float);   //decl^n

void main()      // calling
{
  sum();  (5.1, 1.1)   -or-
}

void sum(Float x, Float y)   // called
{ Float s;
  s = x+y;
  printf("%f ", s);
}
```

(direct) (use)

calling

main: a [5.1], b [1.1]

```
void main
{
  Float a, b;
  printf("enter num ");
  scanf("%f %f", &a, &b);
  sum(a, b);
}
```

sum (free): s [6.2], x [5.1], y [1.1]

o/p ⇒ 6.2000

(91)

* with argument & with Return type

```
int sum (int, int);   // decl"
```

argument.

```
void main ()   // calling
{   ~~sum~~ int x,y;c;
    printf ("Enter value");
    scanf ("%d %d", &x, &y);

    c = sum(x,y);
    printf ("sum = %d\n", c);
}
int sum (int a, int b)   // called
{
    int ~~sum~~ s=0;
    s = a+b;
    return s;
}
```

-OR-

```
return a+b;
```

Return

main: x = 5, y = 7, c = 12

sum: a = 5, b = 7, s = 12

(92) Gate 2019. (1 mark)

```
* #include <stdio.h>
int jumble (int x, int y);   //called
{   x = 2 * x + y;
    return x;
}
int main ()   //calling
{   int x = 2, y = 5;
    y = jumble (x, y);
    x = jumble (y, x);
    printf ("%d\n", x);
    return 0;
}
```

(1) (2) (3) (4) (5) (6) (7)

2*(5+2)

2*12+2

main: x = 2, y = 5; 26, 12

jumble: x = 5, y = 12; 12; x = 12, y = 12

(93) Gate (2) 2000. — 1 mark

```
int incr (int i)   // called
{   static int count = 0;
    count = count + i;
    return (count);
}
main ()   // calling
{   int i,j;
    for (i=0; i<=4; i++)
        j = incr (i);
}
exist
printf ("J = %d", J);
```

only one time hold the previous value

OP.

J = 10

incr: i = 0, 1, 2; count = 0, 1, 3, 6, 10

main: i = 0, 1, 2, 3, 4; j = 0, 1, 3, 6, 10

(94) Gate (3)

```
*  int r()
   {   static int num = 7;
       return num --;
   }
   int main ()
   {   for (r(), r(), r());
       printf ("%d", r());
       return 0;
   }
   exist = 5 2
```

Return + modify

main: 7, 6, 5

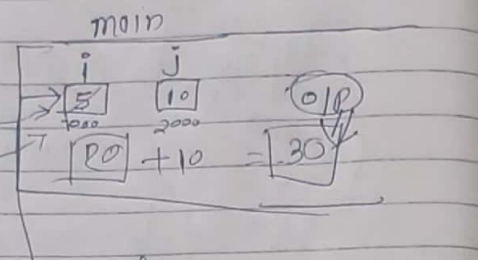
r(): num = 7, 6, 5, 4, 3, 2, 1, 0, -1

(95) Gate - 2016.

```
void f(int *p, int m)
{   m = m+5;
    *p = *p+m;
    return 0;
}

void main()
{
    int i=5; j=10;
    f(&i, j);      // calling
    printf("%d", i+j);
}
```

call by ~~type~~ ref

call by value



(96)

* Passing array as an arguments in function -

(decl) int avg (int [], int);
array

size of (marks)

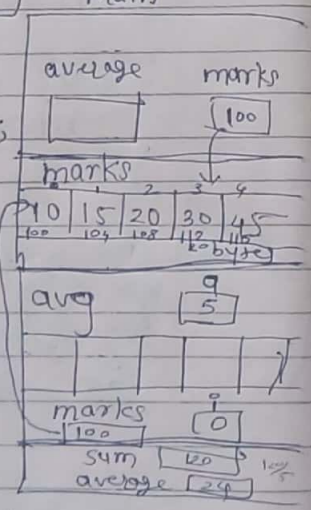
```
void main()
{   int marks[5] = {10,15,20,30,45};
    average = avg(marks,5);
    printf("avg is %d", average);
}

int avg (int marks[], int a)
{   int i, sum=0, average=0;
    for(i=0; i<a; i++)
        {   sum = sum + marks[i];
        }
    average = sum/a;
    return average;
}
```

int average;

(calling)

pointer

(called)



(97)

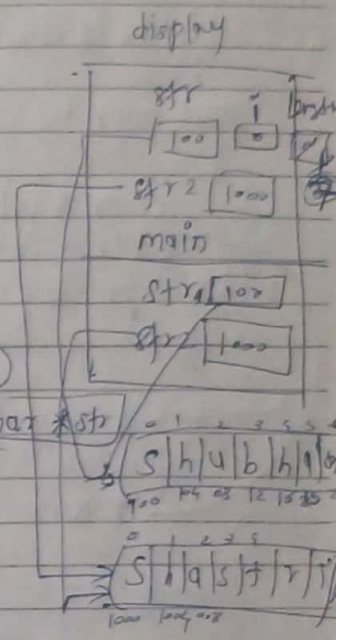
* Passing string as an argument to a function -

```
void display (char [], char [] );
void main()
{   char str1[] = "shastri";
    char str2[] = "Shubham";
    display (str1, str2);
}

void display (char str[], char str[]);
{   //printf("string is %s", str);
    for(i=0; str[i] != '\0'; i++)
        {   length = length+1;
        }
    printf("length of string is %d", length);
    str1[0] = 'S';
    printf("%s %s", str1, str2);
}
```

decl

calling

called



Scanned with CamScanner

(98) Imp

* Return a string from a function.

```
char * display();

void main()
{
    char * str;
    str = display();
    printf("string is %s", str);
}

char * display()
{
    return "shubham";
}
```

Read only memory

dynamic memory allocation

not erasable

-OR-

```
char * display()
{
    static char str[] = "shubham";
    return (str);
}
```

stack memory

lifetime throughout program

memory is vanish

not used

Soln

Program

```
const char * str = "shubham";
```

Compile time error

modify

keep area

not modify

Soln.

problem

display()

str 200

0	1	2	3	4	5	6	7
s	h	u	b	h	a	m	\0

main()

str

100

0	1	2	3	4	5	6	7
s	h	u	b	h	a	m	\0

(99)

* Returning pointer from function

```
int * returnpointer(int[]);

void main()
{
    int *p;
    int a[] = {1,2,3,4,5};
    a = a+2; printf("%d", *a);
    p = returnpointer(a);
    printf("%d", *p);
}

int * returnpointer(int a[])
{
    a = a+2;   // increment pointer
    return a;
}
```

→ 3 (Ans)

int *a

error

constant pointer to change inc/dec (x)

100 + 2 * data type (4)

= 100 + 8 = 112

returnpointer

a 100 108

main

p 100 a 100

const pointer

stack

1 2 3 4 5

100 104 108 112 116

(100) Gate 2010 - Que (fun)

Ans 0 2

```
#include <stdio.h>

void f(int *p, int *q)
{
    p = q;
    *p = 2;
}

int i=0, j=1;

int main()
{
    f(&i, &j);
    printf("%d %d\n", i, j);
    getch();
    return 0;
}
```

(100) (200)

(200)

global variable

i j 25

0 1

100 200

100 = 200

main

f(int *p, *q)

p 100 q 200

200

(101)

nor pointer variable but it take address of fun

pointer ↓ contain data

* Function pointer ⇒ (multi pointers used during fun)

int * ptr ; → (return type of fun) (* pointer Name) (Data type of argument)

address of code

int sum (int, int); — decl

int sum (int a, int b)
{ return a+b;
} // called

~~int (*sum) (int, int) = &sum~~

L → R

int (*ptr)(int, int) = ∑

↳ [*ptr is fun pointer ~~to an~~ which except two argument (int, int) & Return integer value.]

```
# Include <stdio.h>
int sum (int, int); // decl^n
void main()
{
    int s=0;
    int (*ptr) (int, int) = &sum;
        s = (*ptr) (2,3);
    printf("%d", s);
}

int sum (int a, int b)
{
    Return a+b;
}
```

add of code

s [0] Ptr [1000]

[5] O/P

1000 int sum()
1 2
3

sum
a [2] b [3]

(102) appl^n of fun

* Call back Function — passing fun → fun

```
void sum (int a, int b)
{
    printf("%d", a+b);
}
```

5+1 (6) [6]

```
void sub (int a, int b)
{
    printf("%d", a-b);
}
```

(5-1) [4]

```
void display (void (*ptr) (int, int))
{
    (*ptr) (5,1);
}

void main()
{
    display (sum);
    display (sub);
}
```

1000 1000 1 2

done

sum 100

sub 200

ptr [1000] [2000]

Function passing to another function like log

(109)

* Appl^n of function pointer.

Switch case

calculator

add = 0
sub = 1
mul = 2
div = 3

press

```
void add (int a, int b)
{ printf(" %d ", a+b);
}

void sub (int a, int b)
{ printf(" %d \n", a-b);
}

void mul (int a, int b)
{ printf(" %d \n", a*b);
}

void div (int a, int b)
{ printf(" %d \n", a/b);
}

void main()
{
   int ch, a, b;
   void (*fptr)(int, int);
   void (*fptr[10]) (int, int) = {add, sub, mul, div};

   printf("Enter choice");
   scanf(" %d ", &ch);

   printf("Enter a, b");
   scanf(" %d %d ", &a, &b);

   (*fptr[ch]) (a,b);
   // fun^n power
}
```

5

fptr → 100

0	1	2	3
add	sub	mul	div
100	146	200	300

(0) (3)

(104)

* Recursion in c. — (Function called itself directly/indirectly is called recursion).

→ stack overflow problem

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$n = 5 \Rightarrow \boxed{n*(n-1)!}$

120 = 5×4! → 4×3! → 3×2! → 2×1! → 1

n == 0

n <= 1

```
#Include <stdio.h>

void display (int n)
{
   if (n<1) {return;}
   else
   { printf(" %d ", n);
     display (n-1); // Recursion
     printf(" %d ", n);
   }
}

void main()
{ int n=3;
  display(n);
}
```

(3) (2) (1)

(3) (2) (1) (0) (1) (2) (3)

(n=3)

Return back

main → 1x

3	2
n=1	display
display 0	display (1)
n 2	
display 1	display (2)
n 3	
display 2	display (3)

main: n = 3

O/P ⇒ 3 2 1 1 2 3

First move then Return back to stack memory & print the output.

Scanned with CamScanner

(105) Fast method

*

```
int sum (int x)
{
    int s=0
    if (x==-1)
        return x
    else s=x+sum(x-1)
    return s;
}

void main()
{
    int a;
    a = sum(5);
    printf("%d", a);
}
```

main
a = 15

sum(5) = 15
s 0
5 + sum(4) = 15
4 + sum(3) = 10
3 + sum(2) = 6
2 + sum(1) = 3
return 1

1+2+3+4+5

15 o/p

calling

main
a = 15

sum(5) = 15
s = 0

s = x + sum(x-1)
0 = 5 + sum(5-1)
0 = 5 + sum(4)
10
4 + sum(3)
6
3 + sum(2) = 3+3 = 6
5
2 + sum(1) = 2+1 = 3

5+10 = 15
4+6 = 10

sum(int x)
5
3
2
1

Return 1

Return 1

(106)

fun called itself

* Types of Recursion :- (1) direct (2) indirect (3) Tail (4) non-Tail (5) linear (6) Tree

Indirect fun → fun1 ⇄ fun2

```
void main()
{
    printf("%d", fun1(5));
}

int fun1 (int n)
{
    if (n<=1)
        return 1;
    else
        return
        n * fun2 (n-1);
}

int fun2 (int n)
{
    if (n<=1)
        return 1;
    else
        return
        n* fun1 (n-1);
}
```

120

calling

fun1(5) = 120
5 * fun2(5-1) = 120
5*24
4 * fun1(3) = 24
4*6
3 * fun2(2) = 6
3*2
2 * fun1(1) = 2
2*1
return 1

calling

direct

```
fun1()
{
    ____
    ____
    fun2()
}
```

(107)

* Tail & non tail Recursion -

(Tail) ⟹ [after executing Return = nothing left is called]

```
void print(int a)
{
    if (a<1)
        return;
    else
        printf("%d", a);
        print(a/2);
}
void main()
{
    print(10);
}
```

(10) (5) (2) (1) (0)

(10<1) × (5<1) × (2<1) × (1<1) × (0<1)

(10/2) = [5] (5/2) = [2] (2/2) = [1] (1/2) = 0

[10] [5] [2] [1]

print(0) → Return
print(1)
print(2)
print(5)
print(10)
print(10) → main

— [10 5 2 1]

[non - tail] ⟹ after executing Return = (do) left
print(a/2);
add ⇒ printf("%d", a); //

(Same program one line add)

↳ [10 5 2 1 1 2 5 10]

```
void main()
{ int x;
  x = print(10);
  printf("%d", x);
}
```

[4] (o/p)

```
void print(int a)
{ if (a<1) return;
  else
    return;
  1+ print(a/2);
```

(10<1)

non-tail

(108) Gate 2016

*

```
void count(int n)
{
  static int d = 1;
  printf("%d", n);
  printf("%d", d);
  d++;
  if (n>1) count(n-1);
  printf("%d", d);
}
void main()
{
  count(3);
}
```

(only one time initialize)

(3>1) (2>1)

(non tail)

(o/p) ⟹ 3 2 1 1 2 2 3 4 4 4

Hold is previous value

static d [1] → [2]

count(3)
n = [3] d = [1]
count(2)
n = [2] d = [2]
count(1)
n = [1] d = [3]
4

pf(d) [4]
pf(d) [4]

Scanned with CamScanner

(10V)

(109)

template → no memory allocated

→ user defined data type

* Structure in C

— variable of diff data type & under single tag / name.

```
struct student      // decl^n of structure
{
    int rollno;     // member of structure
    char name[20];
    float marks;
};
```

(110)

* Declaring variable / objects of structure

```
struct student
{
    int rollno;
    char name[20];
    float marks;
} s;
};
```

struct student — datatype

tag optional

and used array for decl^n

```
void main()
{   int a;
    // struct name  s,
    datatype        object/variables.
}
```

memory 20 bytes

4 bytes

int	4	4
float		4
char		20

(111)

* Initialization & Accessing of Structure

```
struct student
{
    int rollno;
    char name[20];
    float marks;
};
```

-OR- `} s1={1, "Hello", 90.31};`

s1	
Rollno	1
name	Hello
marks	90

s2	
Rollno	2
name	Hey
marks	93.0

```
void main()     // initialization
{
    // struct student s1={1,"Hello",90.31,93};
    struct student s2 {2,"Hey", 93.};
}
    printf("%d %s %f", s1.rollno, s1.name, s1.marks);
}
```

(112)

* Array of structure.

```
struct student
{
    int rollno;     // member
    char name[20];
    float marks;
};
// struct student s[3];
```

rollno name marks

s[0] s[1] s[2]

s[3]; array

	0	1	2
s	100		

```
void main()
{
    struct student s[3];
```

s[0].rollno
s[0].name
s[0].marks

```
for(i=0; i<3; i++)
    scanf("%d %s %f",
        &s[i].rollno,
        s[i].name,
        &s[i].marks);
}
for(i=0; i<3; i++)
} printf(——)
}
```

Scanned with CamScanner

(113) which point to the add of memory block storing structure

* pointer to structure | structure pointer

```
struct student
{
    int rollno;
    char name[20];
    float marks;
};
struct student s;
void main()
{
    struct student *ptr = &s;

    printf("%d", ptr -> rollno);
```

internal pointer variable

{1, "Shubham"}

rollno	name	marks
1	Shubham	
1000	1004	1024

s [1000] 200 ← ptr [2000]

=1

(*ptr).rollno

arrow

⟹ ok

(114) same as structure

* Union in C. → user defined datatype

→ single memory

→ only 4 byte (ram)

```
union abc
{
    int a;
    char b;
    float c;
};
void main()
{
    union abc U;
    U.a = 1;
    U.b = 97;
    U.c = 9.2;
    printf("a = %d", Ua);
    printf("b = %", Ub);
    printf("c = %", U.c);
}
```

[97 | 9.2] a, b, c

⟹ not used

(115)

* Typedefined structure

```
typedef int integer;
main()
{   int a;
    integer a,
}
```

datatype → new datatype defn

typedef int integer

type | datatype | synonym

```
typedef struct student
{   int a;
    char b;
    float c;
} student;
void main()
{
    student s = {10, 'H', 9.5}
    printf("%d, %c, %f", s.a, s.b, s.c)
}
```

datatype

New Name

synonym for datatype name

(116) * Structure padding in C → memory hole | alignment

(CPU = speed increase)

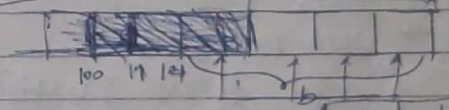
(2 CPU cycle)

compiler do automatically

```
struct abc
{
    char a;
    int b;
    char c;
} s
```

(8 byte)

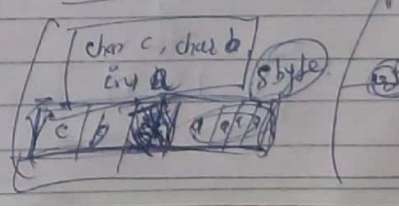
→ with structure pack → (12 byte)



word addressable

32 bit = 4 byte

64 bit = 8 byte



(117)

Scanned with CamScanner

(117)

* Structure Packing in C

```
# pragma pack (1)
struct abc
{ char a;
  int b;
};
main()
{ struct abc s;
  printf("%d", sizeof(s));
}
```

→ packing features turn on → (padding not occurs)

char a; — (1) + 3 (padding add)

int b; — (4)

(8 byte)

(5 byte)

(118) * Problem on structures in c

```
struct question
{
  int a;      ← 1
  float b;    ← 10
  int c;      ← 15
};
void main()
{ struct question q = {b=1; c=10, a=15};
  printf("%d %f %d", qa, qb, qc);
}
```

(1)

⇓

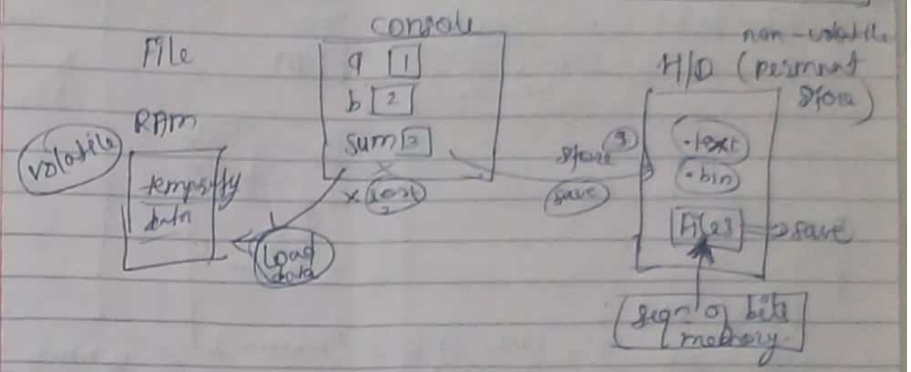
1, 10.00, 15.

(119)

* File handling in c → type → text (.txt), Binary (.bin)

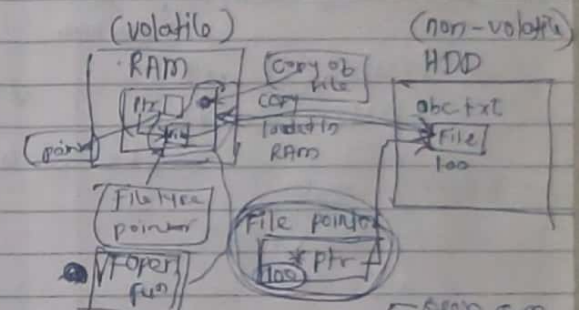
datatype: FILE

operation of file: create + open + Read + write + close



(120) * FILE *ptr

```
typedef struct
{
  *ptr
}FILE;
```



FILE *fp

mode → ① r - read
② w - write
③ a - append
④ r+ = read/write
⑤ w+ = write/read
⑥ a+ = all.

open fun ⇒ fopen("filename", "mode")

fopen(abc.txt, r)

⇒ fclose(fp); // close fun

operation ⇒
- fprint
- fscanf
- fputc
- fgetc
- fputs
- fgets

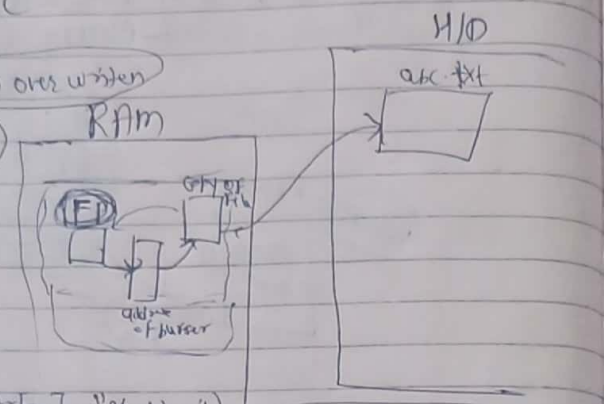
(121)

* Write into a file in C

Case

① File exist : contain over written

② Not exist (It create new file)




```
main ()
{
    FILE *FP;
    char ch = 'a'; char str[20] = "shubham";
    FP = FOpen ("abc.txt", "w");
                              ↑ write mode
    Fputc (ch, FP);
    Fputs (str, FP);
    // string length
    Fclose (FP);

    // Formatted o/p fun ⇒ Fprint (FP, "%c %s", ch, str);

    = Fclose (FP);  end.
```

writing.

Fputc ⇒ char in file

Fputs ⇒ string in file

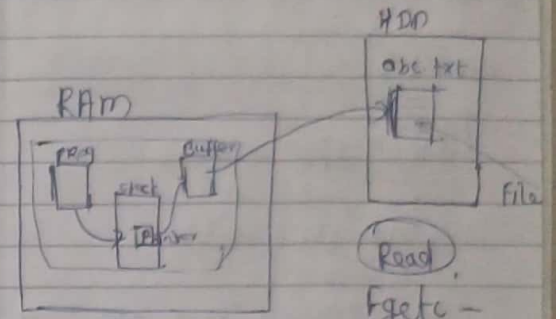
Fprint ⇒ o/p screen to file write

```
{For (i=0; i != strlen(str); i++,
        Fputc (str[i], FP);}

Cond^n.
if (FP == Null)
    printf ("no file")
    exist(1);
```

(122)

* How to Read from a file in C -

"r"




```
main ()
{ FILE *FP = NULL;
    char ch;
                    Read mode
                      ↓
    FP = FOpen ("abc.txt" "r");
        if (FP == NULL)
            { printf ("error");
              exit(1);
            }
        while (! Feof (FP))
        { ch = Fgetc (FP);
          printf ("%c", ch);
        }
    Fclose();
}
```

-OR-

Read

Fgetc -

Fgets -

(Feof (end of file))

String ⇒ char s[30];

```
Fgets (str, 7, FP);
        ↑ length of string
        ↑ File name
printf ("%s", str);
```

Full string Print of file (Use case)

```
while (! Feof (FP))
    { Fgets (str, 7, FP);
      printf ("%s", str);
    }
```

(123) Storing previous data + new data

* Append mode in File Handling.

```
main()
{
  FILE *FP = NULL;

  char str[50];

  FP = Fopen("abc.txt", "a");

  if ( FP == NULL)
     { printf("error");
       exit(1);
     }

  printf("Enter the content to append");
     gets (str);
  fprintf (FP, "%-s", str);   // add/modify
  printf("successfully");

  Fclose(FP);
}
```

file pointer

string

Fputs (str, FP);

abc.txt

Shubham
Shastri

Rewind (FP); → cursor starting

add

(124) special mode

* r+ mode in File Handling

↳ Read + writing ⇒ doesn't exist: not create new file.

```
main()
{
  FILE *FP = NULL;
  FP = Fopen ("abc.txt", "r+");
     if ( FP == NULL)
        { printf("error");
          exit (1);
        }
     Fputc ("z", FP);
     Fputs ("shubham", FP);

     Fclose(FP);
}
```

abc.txt

hey, I am shubham
Shubham
z
(modify existed content)

abc.txt

(125) * w+ mode in File handling.

→ Reading + writing
→ Create new file (doesn't exist file)
→ File exist then erase the privous content.

```
FILE *FP = NULL;
char str[20];
FP = Fopen ("abc.txt", "w+");

  if ( FP == NULL)
     { printf("error");
       exit(1);
     }
  Fputs ("Shastri", FP);
  while ( !feof (FP)) {true = 1} 0
     { ch = Fgetc (FP); print ("%c", ch); }
```

abc.txt

Shubham
Shastri
FP

check

(126)

* a+ mode in file handling
↳ Reading / appending (write at end of file) → create a new file (doesn't exist)

```
FILE *FP = NULL;
    char ch;
FP = FOpen("abc.txt", "a+");
    if (FP == NULL)
        { printf("error");
          exit(1);
        }
  fputs("Shastri", FP);
  rewind(FP); // cursor
  while (!Feof(FP)) // Read
  {
    ch = fgetc(FP);
    printf("%c", ch);
  }
  fclose(FP);
}
```

abc.txt

Shubham Shastri

(used) → [end to beginning]

→ (Shubham)

(To move a cursor on specific pointer / position)

(127) * FSeek( ) in file handling

FILE, long int, int origin

fseek(file pointer, offset, position)

SEEK_SET (beginning)
SEEK_CUR (current)
SEEK_END (end)

```
{
  FILE *FP = NULL;

  FP = FOpen("abc.txt", "r");
    if (FP == NULL)
      { printf("error");
        exit(1);
      }

  Fseek(FP, 7, SEEK_SET); // beginning 0-7
    ch = fgetc(FP);
    printf("%c", ch);

  Fseek(FP, -2, SEEK_CUR); // cursior
    ch = fgetc(FP);
    printf("%c", ch);

  Fseek(FP, -2, SEEK_END); // end
    ch = fgetc(FP);
    printf("%c", ch);
```

abc.txt

Shubham Shastri

(Read)

→ [S]

→ [m]

→ [T]

Scanned with CamScanner

(128)

* ftell() fun in file handeling
  ↳ name suggest (position of file pointer)
  ↳ starting of line

Return type
long int ftell(file *pointer);

abc.txt

0	1	2	3	4	5	6	7	8
s	h	u	b	h	a	m	_	sha

ftell(a) = 5

```
FILE *FP = NULL;
char ch; int pos;
char str[50];

FP = Fopen("abc.txt", "r");  //read
if (FP == NULL)
    { printf("error");
      exit(1);
    }
```

or →

```
Fseek(FP, 0, SEEK_END);
printf("%d", ftell(FP));
```
= 22

```
pos = ftell(FP);
printf("%d", pos);  ⇒ 0

Fseek(FP, 5, SEEK_SET);
{ ~~ch = fgets~~
~~Fclose(FP)~~;  printf("%d", ftell(FP));  ⇒ 5
}

~~fgetc~~ ch = fgetc(FP);
printf("%c", ch);
printf("%d", ftell(FP));  6

fscanf(FP, "%s", str);
printf("%s", str);
printf("%d", ftell(FP));  = 22
```

(129)

* rewind() fun in file handling.
  ↳ Set a cursor/pointer to begenging (starting)

abc.txt

shubham shastri

shastri

shubham shastri

```
{ FILE *FP = NULL;
  char ch;

  FP = Fopen("abc.txt", "r+");
  if (FP == NULL)
      { printf("error");
        exit(1);
      }
  ~~Rewind~~ rewind(FP);  //
  Fseek(FP, 0, SEEK_SET);  //begi.

  while(!Feof(FP))
      { ch = Fgetc(FP);
        printf("%c", ch);
      }
  //rewind(FP);  // set a cursor at begining

  Fclose(FP);
}
```

Scanned with CamScanner

line + char → H.W

* Program to count no. of ~~times~~ in file.

```
{
  FILE *fp = NULL;
    int count = 1;
    char ch;

  fp = fopen("abc.txt", "r");
    if( fp == NULL)
        { printf("error");
          exit(1);
        }
  while((ch = fgetc(fp)) != EOF)   //end of file
    {
      if (ch == '\n')
        { count ++;
        }
    }
  fclose(fp);
  printf("%d", count);   ⇒ 2
}
```

line | abc.txt
1 | Hi
2 | Hello.
3 |
4 | Shubham
5 |
6 |

* Copy content of a file into another file :

```
FILE *Fptr1, *Fptr2;
  char c;

• Fptr1 = Fopen("abc.txt", "r");
    if (Fptr1 == NULL)
        { printf("error");
          exit(1);
        }
• Fptr2 = Fopen("destination.txt", "w");
    if (Fptr2 == NULL)
        { printf("error");
          exit(1);
        }
  while((c = Fgetc(Fptr1)) != EOF)
    {
      Fputc(c, Fptr2);
    }
  printf("done");

  fclose(Fptr1);
  fclose(Fptr2);
}
```

abc.txt
1 | Hi
2 | Hellow

Fptr1 → 2

Read, fgetc, copy

destination.txt
Hi
Hellow

FPTR2 → 3

write, Fputc

EOF = 0

done

(132)

✓ Run time

Introduction to Dynamic memory allocation in C.

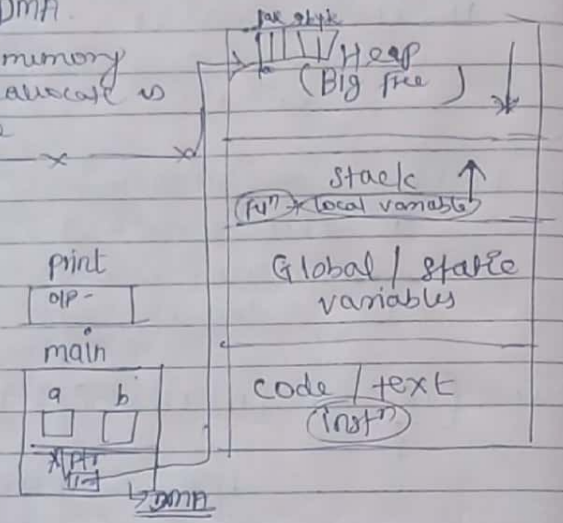
- Static memory allocation at [compile time] (conv.) obj code
- Dynamic memory allocation at Run time

fu^n → [malloc], [calloc], [realloc], [free]

→ appl^n of pointer to DMA

→ Reused memory allocate / deallocate in Run time

```
main()
{
  int *p;

}
Free(*p);
```



Static memory allocation	Dynamic memory allocation
- performed at static / compile time	- Performed at dynamic or Run time.
- Assigned to stack	- Assigned to heap.
- First in last out.	- No particular order of assignment.
- Size must be know at compile time	- Size may be unknown at compile time.
- memory size cannot change	- Size can be changed

(133)

[structures] ↑ and (memory allocation)

* Dynamic Memory Allocation using malloc() fu^n.

```
int *ptr;
void * malloc (size_t size)
```

(type cast) (generic pointer) (unsigned int)

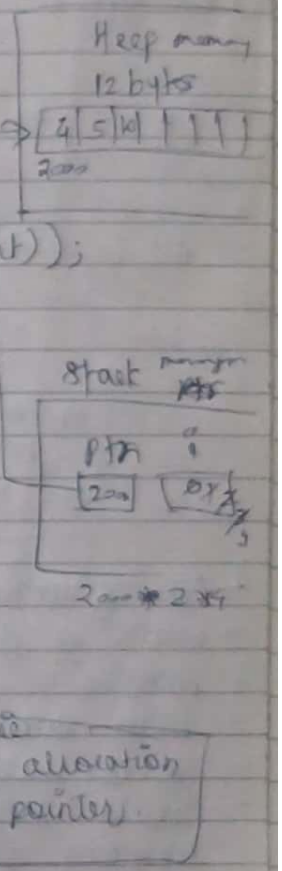
```
ptr = (int *) malloc (sizeof (int))
```

```
int main()
{
  int n, i, *ptr;
  printf("Enter total no of values");
  scanf("%d", &n);
  ptr = (int *) malloc (n * sizeof (int));

  printf("Enter values");
  for (i=0; i<n; i++)
    { scanf("%d", (ptr+i));
    }

  printf("The entered values are");
  for (i=0; i<n; i++)
    { printf("%d", *(ptr+i));
    }

  Free (ptr);
}
```

n = [3]

3 * 4 = 12

(scan)



→ dynamic memory allocation using pointer.

Scanned with CamScanner

(134)

* calloc () fun — Dynamic memory Allocation.
  - → Contiguous allocation
  - → Built in fun ⇒ stdlib.h
  - → ~~Dr~~ dynamically allocated multiple blocks of memory & each blocks is of same size.

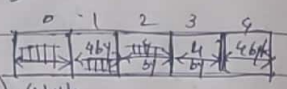
void * calloc (size_t n , size_t size)

- void * ↓ (Return type)
- n ↓ (no. of blocks)
- size ↓ (size of each blocks)

int *ptr; ← static memory allocation

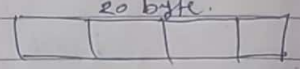
ptr = (int *) calloc (5, sizeof (int));

[ini ⇒ 0]



ptr = (int *) malloc (5 * size of (int));

(ini ⇒ garbage)



int = 10 value
free
float to value

[code same as malloc]

malloc ()	calloc ()
- No. of argument is 2	- is 1.
- calloc is slower than malloc	- malloc faster than calloc.
- High time efficiency	- lower time efficiency.
- gives garbage value	- initialize with zero.
- Create single block of memory size	- assign multiple blocks memory for a variables.

(135) realloc

* realloc () — dynamic memory allocation (DMA)
  - → reallocation of memory
  - → Resize (incr./decr.)
  - → without losing the previous content

void * realloc (void * pointer, size_t size);

- void * ↓ Return pointer
- pointer ↓ previous memory
- size ↑ new size.

int *ptr;

ptr = (int *) malloc (5 * sizeof (int));

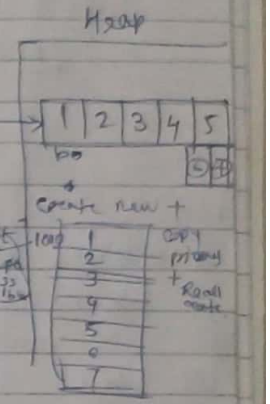
(2add) ~~2x~~ (if possible)

ptr1 = (int *) realloc (ptr, 7 * sizeof (int));

→ (NULL)

Ptr, ptr1
↓ ↓
100 100

malloc(100)



#
```
int *ptr, n, i;
printf("enter value of n");
scanf("%d", &n);

ptr = (int *) calloc (n, sizeof (int))
printf("enter values");
for(i=0; i<n; i++)
  { scanf("%d", (ptr+i));
  }
printf("enter updated");
  { scanf("%d", &n),

int *ptr1 = (int *) realloc
            (ptr, n* sizeof (int));

printf("values are");
for(i=0; i<n; i++)
  {
    printf("%d", *(ptr1+i));
  }

Free (ptr1);
}
```

Scanned with CamScanner

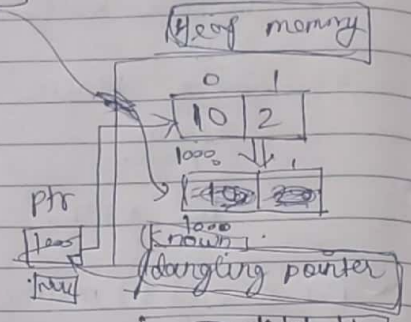
(manually free)

(136) (stdlib.h) always Free memory used

* free() – dynamic memory allocation (DMA).
  - → Release the dynamic memory allocation
  - → ~~free(~~ [void free(pointer)]

```
int *ptr;
ptr = int(*) malloc (2 * sizeof(int))
 =
 =
Free (ptr); // ⇒ undefined behaviour
ptr = NULL;
```

(after free)



(forgot heap memory free)

(137)

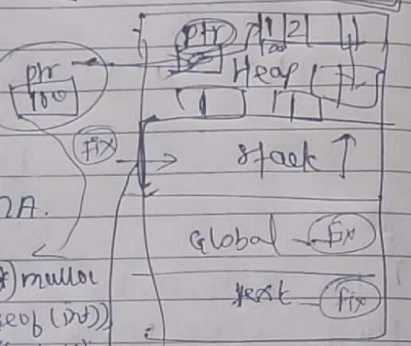
* memory leak in [C/C++]
  - ↳ Improper used of heap / DMA.

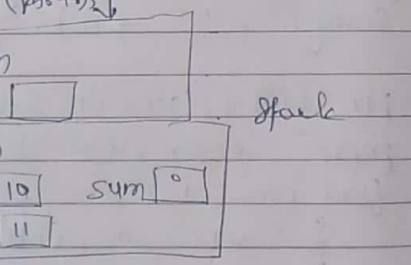
#

```
int sum()
{ int sum = 0;
  int a = 10, b = 11;
  sum = a + b;
  printf("%d", sum);
}
int main()
{ int ch = 1;
  while (ch);
  { sum(1);
  printf("continue?");
  scanf("%d", &ch);
}
```

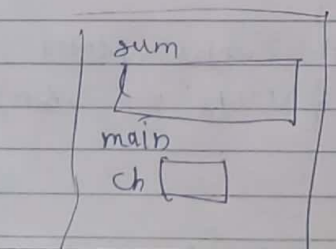
static (func) [int sum()]

```
{ int *ptr = (int *) malloc
  (2 * sizeof(int))
  sum = (*(ptr+0) + *(ptr+1));
```





free



dynamic

garbage

(138) Enum ← user defined data type

[local + global]

[#] – pre process phase → global scope

* Enumeration in C (enum)
  - ↳ to assign names to integral constant.

ex. define max 50; ⇒ (max)

```
enum days {
    sun,
    mon, };
```

```
Struct student
{
 =
 =
};
```

(Next value = previous value + 1)

#

```
enum week_days { sun, mon, Tue, wed, Thus, Fri, sat };
                 0   1   2   3   4    5    6
int a;
enum week_days today, today1;
today = Tue;      // one time one set value
today1 = Fri;
printf("%d", today); ⇒ Tue 2
```

(previous value then continue)

→ with same scope the duplicate value not allowed in enum.

→ only integral values are allowed (int)

→ switch case ⇒ switch ( )

```
case sun : pf("today is sun");
case mon : ——— " ——— " ———
```

(H.W)

#

gender discount

```
if  F → 50%
    M → 25%
    O → 60%
```

```
if (costomer == female)
{
 —  —
}
```

Scanned with CamScanner

enum gender { male, female, other };

enum gender g;
g = male

if (g == male)

(139) → int value default 0

* Enum - Interview questions.

~~typeof~~
typedef enum {
male, 0
female, 1
} gender;

int main()
{ gender var = female;
printf("%d", var);
return 0;
}
⇒ 1

```
# include <stdio.h>

enum Xenum { C, CPP, Java };
/ enum Yenum { Xenum }
main ()
{ enum Xenum var;
/ enum Yenum var;
printf("%d", sizeof(var));
return 0;
}
```

0 1 2

at a time one value

o/p ⇒ 4 ⇒ bytes (int)
⇒ 4 byte

(140)

* Storage classes in C

int a = 10;
type, name, value
storage class

Storage class
① default value it take
② location (RAM)
③ scope (local/global)
④ lifetime (till when)

→ Types ⇒ - auto
- register
- extern
- static

auto int a;
storage class

* Scope = Block → { = }
Fun / method → void add { = }
program →

```
# include <stdio.h>
  int a=50;  } — [program scope]
int main()
{   // int a=10;      fun();
    {
      int a = 20;
      printf(" Inside Block %d", a);
    }
    {
      printf("%d", a++);
    }
    printf("%d", a);
    return 0;
}
void fun()
{   int a=30;
    printf("%d", a);
}
```

[Block scope]

B.S 2

int	O/P
30	30
20	20
10	50
11	51

global value

10 +1

50+1 = 51

11

51

Fun scope

141

* auto storage class :-

141 ↑ automatic

* [auto] storage classes

① auto, Register, Static, External

② Default value, Scope ③, location, lifetime → Program, Block, Function

* Automatic (auto) → [only local variable]

→ default value ⇒ garbage value

→ Scope → Block / Function

→ Lifetime → within the block

→ Location → stack memory (RAM)

```
int main()
{
  auto int x=10; pf("");  -or- [int x=10]
  {
    auto int x; pf("");
  }
}
```

Block scope

142

Scope [counter variables, loop variables] → not used in pointer

[Global variable not allowed]

* Register Storage classes → [default value] ⇒ garbage

→ Scope → Block / fun

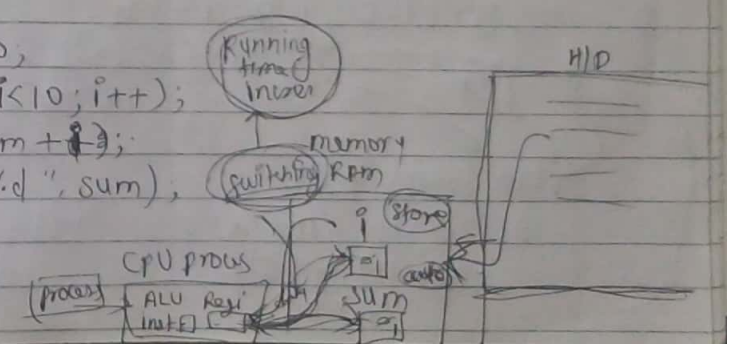
→ Lifetime → within the block

→ Location → CPU Register

[register int a;]

```
int main()
{ register int i, sum=0;
  for(i=0; i<10; i++);
  { sum = sum + i;
  printf("%d", sum);
}
```

[fast access] → CPU Reg



(143) Imp

initialize value only one time then save previous value then add/sub/any fn

* Static storage class

- default value ⇒ '0' zero
- Scope ⇒ with_in block
- lifetime ⇒ till end of program
- location ⇒ RAM → global / static

```
#include <stdio.h>
void display();

void main()
{   display();   //calling
    display();   //calling
}   printf("x=%d", x)

void display()
{   static int x;=0
    x += 10;   // x=x+10;

    printf("\n x=%d", x);
}
```

⇒ lifetime → till end of program values

x: 0, 10

undeclared error

Retain in previous value

10 + 10 = 20

display: 10 ① ; display2: 20

Return

```
void display();

void main()
{ int i;
  for (i=0; i<3; i++)
    display();
}

void display()
{ static int x=5;
  int y=5;
  x++;
  y++;
  printf("x=%d", x);
  printf("y=%d", y);
}
```

x = 8 6 = y

i	x	y
0	5	5
1	6	6
2	7	7
3	8	8

(144)

only Global variables to decl^n of global / fun

decl^n ⇒ just told data type no space allocation

defn → space allocation

* Extern storage classes

access variable to another file

- default value → zero
- Scope ⇒ Global
- location → RAM
- lifetime ⇒ till the end of the program

File 1.c
```
int x=10;
```
⇒ 10

File 2.c
```
extern int x;
// decl^n.
```

⇒ linking two files

Reuse

```
#include <stdio.h>
int a=10;   // global variable
int main(){ extern int a;
  printf("%d", a);
  fun1();
  fun2();
}

void fun1()
{ int a=2;
  a++;
  printf("a=%d", a);
}

void fun2()
{
  printf("Hello");
}
int a=10;
```

decl^n of global variable value

printf("%d", x); ⇒ 10

a 10

printf("%d", a); 10

a++; 2+1

printf("a=%d", a); ⇒ 3

Hello

Scanned with CamScanner

File include

File1.c

```
# include <stdio.h>
# include "support.c"
int x=10;
extern void display();

void main()
{
    display();
}
```

support.c

```
# include <stdio.h>

void display()
{
    extern int x; x++,
    printf("Hellow from supported file");

    printf("x = %d", x);
}
```

x=10 → 10 → 11

all program file folder

⇓

[any program + [extern] used access file]

Q.1) # include <stdio.h>

```
extern int a; // global variable assign = 0
int main()
{ printf("a = %d", a);
  return 0;
}
```

O/P ⇒ a = 0

(145) Q.2

```
# include <stdio.h>
int fun()
{ static int num=16;
  return num--;
}

int main()
{ for (fun(); fun(); fun())
    printf("%d", fun());
  Return 0;
}
```

num = 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

O/P: 14 | 11 | 8 | 5 | 2

⇒ 14 | 11 | 8 | 5 | 2

(146)

* Pre-processor → [text substition tool]

Step in compile process

↳ Program (not a part of compiler)

cmd ↳ # include <stdio.h> ← standard fun^n defn

define PI 3.14

File.c → Pre processor # → stdio.h

if
else
pragma

— expanded source code.

— substituted the complete file to compiler

[# include "stdio.c"] → [# include <stdio.h>] contain → source code

(147) Gate = 2012

* Q.1

```
int a, b, c = 0;

void prtFun(void);
int main()
{
    static int a = 1;
    prtFun();
    a++ a += 1,
    prtFun();

    printf("\n %d %d", a, b);
}
```

a: 1 → 2

```
void prtFun(void)
{ static int a = 2;
  int b = 1;
  a += ++b;
  // a = a + ++b; 2+2
  printf("\n %d %d", a, b);
}
```

a: 2 → 4, b: 1

O/P: 4 2

auto. reg

4	2
4	2
2	0

4 2
6 2
2 0

extern ⇒ check outside

```
int main()
{
  extern int i;   (X)

  printf("%d", i);
  { int i = 10;
    printf("%d", i);
  }
  return 0;
}
```

Linker error

```
int i = 5;
int main()
{
  extern int j;
  printf("%d %d", i, j);
  int j = 10;//not access
  return 0;
}
```

i: 5, j: 5

⇒ 5, 10

Search outside

```
main()
{
  register int data = 10;
  int * pisumData = NULL;
  pisumData = &Data;
  * pisumData = 5;

  printf("%d", *pisumData);
}
```

(cpu regi)

(* not store)

⇒ error

---

(148) macro in c

* pre-processor cmd # define & # undef in c

constants (const)

```
#define PI 3.14159
int main()
{
  int r = 5;
  printf("area of circle
         = %f", PI*r*r);
  return 0;
}
```

3.14

⇒ 3.14159×5×5

```
#define A 10
int main()
{ int x = A;
  printf("%d", x);
```

⇒ 10

```
#define MUL (a,b) a*b
int main()
{ printf("%d", a*b
  return 0;
  MUL(2,3));
}
```

a*b

```
#define NAME "Shubham shastri"
int main()
{
  printf("%s", NAME);
}
```

⇒ Shubham shastri

(5-2*7+4)

a b

(5-14+4)

a = -5

#undef → stop working define macros

(149)

* Predefined macros in C. [conditional cmd]

mmm-DD

```
printf("DATE %s", __DATE__);
printf("Time %s", __TIME__);
```
↳ HH:mm:ss

```
printf("FILE %s", __FILE__);
printf("LINE %s", __LINE__);
```

#ifdef # elif (include or
if # else ⟹ skip before execution)

—x—x—x—x—x—

```
#ifdef MACRO
    code
#endif
```

(when true) ↓ out

(90)

—x—x—x—x—x—

(150)

* [DONE]

* ROS – Robot operating system [used C++/python]

⇓ used for

Robotics system. (software)

- set of tools and a framework.
- designed for robots, providing necessary infrastructure to manage.
- ROS ⟹ helps — ~~wh~~ write a software ⟹ modular way ⇓ making to Reuse code & collabrate with other

Ros Supports modular software design ⟹ Nodes

① Nodes → Individual programs that perform specific tasks [e.g. controlling a motor, processing sensor data]

Ros system ⟹ made ~~of~~ up of — [Node, Node, Node] interacting

② TOPICS -

- Named buses over which nodes exchange message.
- Nodes can "publish" message to a ⇓ Topic or "subscribe" to a topic to ~~Ros~~ ⇓ Receive message.
- This is how diff. parts of the Robot communication.

③ Messages.

- The data exchanged between nodes.
- ROS has standard message types for common data [e.g. sensor readings, motor commads].

Scanned with CamScanner

④ Services -

- Allow nodes to make requests and receive responses. useful for actions that require feedback.

⑤ parameter server.

- A central storage for configuration parameters that nodes can access.

Step-by-Step

① Installation

② workspace setup → This involves setting up a directory structure & environment variable

③ Package creation → use Ros tools to create a new package for our robots functionality.

④ Node Development → write nodes (programming) - define how our nodes will intract with other nodes using topics, services/action.

⑤ Building → cartkin_make / colcon build tools to compile our Ros package

⑥ Running → use - [rosrun] ⇒ cmd → start nodes
[roslaunch] → start multiple nodes.

⑦ Visualization → use tool ⇒ RViz → visualize Robots sensors, models & data.

Scanned with CamScanner