C-LANGUAGE

- Intro programming in C:

(1) All c programs must ~~a~~have a function in it called main.

(2) Execution starts in function main.

(3) comments start with /* and end with */.
or //.

(4) All c statement must end in a semicolon (;)

(5) the #include <stdio.h> statement Instructs ~~of~~ the C Compiler to Insert the entire contents of file stdio.h in its place and compile the resulting file.

- C-Tokens: The smallest individual unit are known as c tokens.

C-haves 5-types of Tokens:
- → Keywords. (break, char, int, continue, default, do..)
- → Identifiers. (user defined word: int money;)
- → Constants (100 is integer constant, a is character constant)
- → Operators. (AO-(+,-,*,/), LO-(&&, ||, !))
- → Special symbols. (separators- (, ; , " "))

- Data types:
  - → primary — (int, char, float, double)
  - → Derived/user defined — (array, string, structure, union)

- Different types of modifier with their Range :

Types of Modifier	Size (in byte)	Range of values
int	2	-2^{16-1} to $+(2^{16-1}-1)$
signed int	2	-2^{16-1} to $+(2^{16-1}-1)$
unsigned int	2	0 to $(2^{16}-1)$
short int	2	-2^{16-1} to $(2^{16-1}-1)$
long int	4	2^{32-1} to $(2^{32-1}-1)$
float	4	$-(3.4E+48)$ to $+(3.4E+48)$
double	8	$-(1.7E+308)$ to $+(1.7E+308)$
char	1	-2^{8-1} to $(2^{8-1}-1)$
unsigned char	1	0 to $(2^{8}-1)$

- Types of operators :

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

(2) Assignment operators (=, +=, -=, *=, etc)

(3) Relational Operators (~~dek~~ <, <=, >, >=, != , ==)

(4) logical operators (&&, ||, !)

(5) Bitwise operators (&, |, ~, ^, <<, >>)

(6) special operators (sizeof(), termary operator)

(7) Opinter Operators (* - value at adress, & - Adress ~~of~~ operator).

- Type Conversion :

(1) Implicit Type Conversion : There are certain cases in which data will get automatically converted from one type to another.

Example :
```
main(){
float z;
int x=10;
char y=a;

x= x+y
z=x+1.0;

print ("x= %d" ,z =%f", x,z);
return 0;
}
```

Output : x= 10+97 = 107 (Ascii value of 'a' is 97)
z = 107+1.0 = 108.000000

(2) Explicit Type Conversion : (user defined).

Example:
```
int main(){

double x =1.2;

int sum=(int) x+1
pf(" sum = %d", sum);
return 0;
}
```

output : sum= 2

• Expression –

① lvalue :

→ Expression that refer to a memory location are called "lvalue" expression.

→ An lvalue may appare as either the left-hand or Right-hand side of an assignment:

$\overset{\text{lvalue}}{b} = 10$

$a = b$

② rvalue :

→ The term rvalue refers to a data value that is stored at some address in memory.

→ can't appear on the left hand side.

• C variable types:

→ Local variable.
→ Global variable.

ex:

```
# include <stdio.h>
int x=10;  ——→ Global variable.
void main() {
   int a~~bee~~ =5; }
   int c ;        } ——→ Local variable.
   c = a+x ;
   printf(" sum = %d', c)
  }
output : 5+10 = 15
```

- Operators in C :

	Operators precedence	Associativity
/	() [] → .	Left to Right
U	!, ~, ++, --, +, -, *, &, (type), size-of	Right to Left
A	*, /, %	Left to Right.
	+, -	"
S	<<, >>	"
	<, <=, >, >=,	"
C	==, !=	"
	&	"
B	^	"
	|	"
	&&	"
l	||	"
	?:	Right to Left
a	=, +=, -=, *=, /=, %=, &=, ^=,	Right to Left
	|=, <<=, >>=	~~Left to Right.~~
c	,	Left to Right

(to remember)

- Format specifiers :

10 %d → prints as decimal number.

------ %6d → prints as decimal number, atleast 6 characters wide.

%f → Prints as floating point.

%6f → prints as floating point, atleast 6 character wide

--- % .2f → prints as floating point, 2 characters after decimal point.

---------- % 6.2f → Print as floating point, at least 6 wide and 2 after decimal - point.

%c → print as ascii character.

%lf → format specifiers for double.

- Charracterr Input and output :

getcharc() :- it rreads the next input characterr from a text stream and returnns that as its value.

c= getcharc()

the variable c contains the next Charracterr of input.

putcharc() :- putcharc prints as characterr each ~~it~~ time it is called.

example :

/* cppy input to output */

```
#include <stdio.h>
void main(void){
    int c;
    c=getcharc()
    while (c != EOF){
        putcharc(c);

        c=getcharc();
    }
}
```

Linux commands

① vi filename.c

windo
:wq → enterr

② gcc filename.c
(complie this file)

③ ./a.out
(to rrun)

```
$ ./a.out <infile.> outfile.
```

- **Storage classes In c :** we have four types of storage classes in c :

(i) Auto storage class.
(ii) Register storage class.
(iii) static storage class.
(iv) Extern storage class.

Storage Class	storage location	Default initial value	Declaration location	~~so~~ scope (visibility)	Lifetime (Alive)
auto	Memory	garbage	Inside a function / Block	within the function/block	Until the function/block compute
register	CPU register	garbage	"	"	"
static (local)	Memory	0	Inside the function/Block	"	Until the program terminates
static (global)	Memory	0	outside all functions	Entire file in which it is declared	Until the program tarminates
extern	Memory	0	outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates

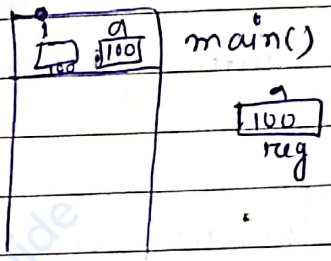
• examples-(on storage classes):

①
X
```
int main(){
register int i=10;
  int *a = &i;
  printf("%d", *a);
  return 0;
  }
```
→ i value stored in register. but register ~~a~~ have no address. so compiler error can be occure in this case.

O/P: ~~w~~ error.

②
✓
```
int main() {
  int i=10;
register int *a = &i;
printf(" %d", *a);
return 0;
}
```
O/P: 10.

activation record.

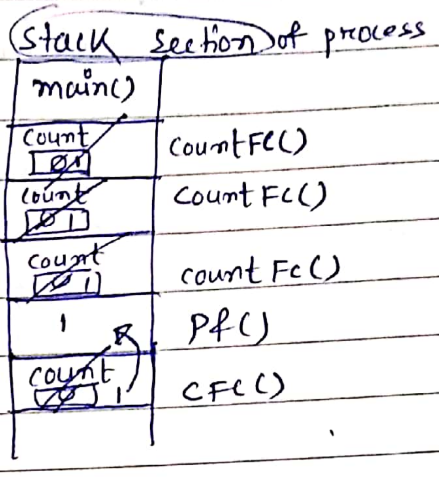


③
X
```
int main() {
  int i=10;
register static int i=10;
pf(" %d",i);
return 0;
}
```
→ storing the value of i ~~and~~ in two diff places ~~also~~ ~~o/p~~ it not possible.

O/p: compliler error

④ ①

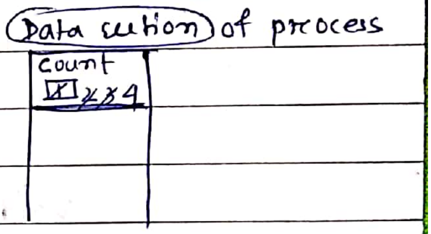
```
int countFunctionCall (void)
   {
 [Auto] int count = 0;
     return ++count;
   }

int main() {

1 CountFunctionCall();
2 CountFunctionCall();
3 countFunctionCall();

4 printf("%d times function is called", count Functioncall());

return 0;
}
```



O/P : 1   1 times function is called.

④ ②

```
int countFunctioncall (void) {
 [static] int count;
   return ++count; }

int main() {
 countFunctionCall();
 countFunctionCall();
 countFunctionCall();
printf("%d times function is called", CountFunctionCall);
 return 0;
}
```

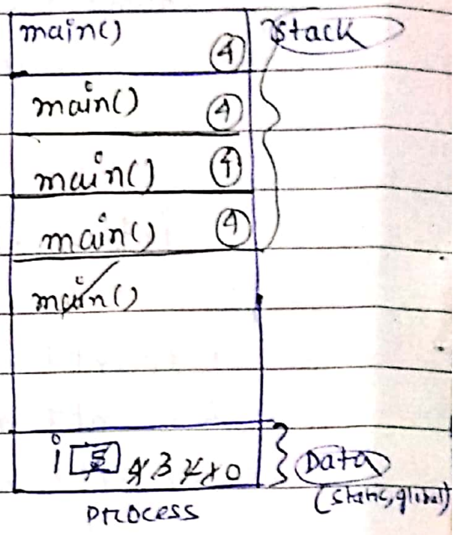


O/P : 4   4 times function is called.

⑤ What is the output of the following program?

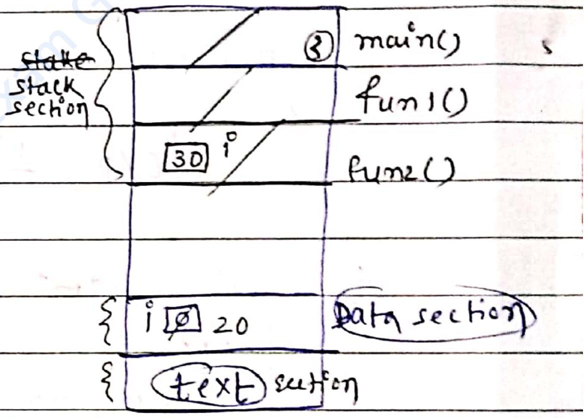
```
#include<stdio.h>
  int main(){
1  static int i=5;
2    if(--i) { 3main();
   4printf("%d",i);
    }
  }
```

O/P: $\boxed{0000}$



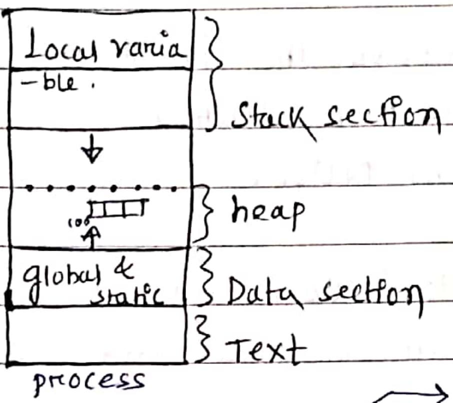
⑥

```
# include<stdio.h>
 int i;
 void fun1(){
   i=20;
   printf("%d", i);
  }

 void fun2(){
   int i=30;
   printf("%d",i);
  }
  int main(){
 1 fun1();
 2 fun2();
 3 return 0;
  }
```



O/P: 20 30

• Storage Management: #Include <stdlib.h>



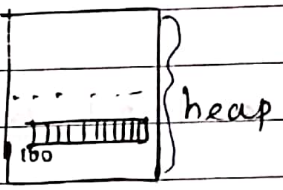
① void *malloc (size_t n) → how many element want to store.

→ Unsigned data-type Size atleast 16 bits.

ex: void *malloc (sizeof(10))

→ It allocate in heap of 10 bytes, and return the pointer (starting address) of the ~~spaca~~ allocated space.

```
int *i;
i = (int *)malloc (sizeof(int));
i++;
```

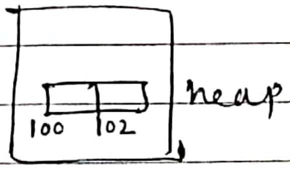


② void *calloc ($\overset{2}{\text{size\_t n}}$, $\overset{2}{\text{size\_t size}}$)

size_t n → how many element you want to store

size_t size → what is the size of each element

→ malloc and calloc always gives space in contiguous menare.



pointer of present space

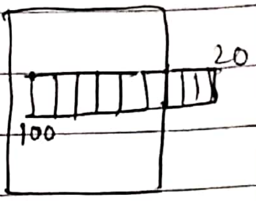
100   20 how ma byte you want to increase.

③ void *realloc (void *ptr, size-t size)

→ it used to increase the size of space.

→ if space is ~~#~~ not available then it returns NULL.

~~[illegible]~~



ex: void *realloc (100, sizeof(20))

→ pointer of allocated space and increase upto 20 byte.

100

④ void free (void *ptr)

→ free the present allocated space by passing the point of the location.

→ to avoide memory lick problem ~~w~~ we use free.

- Input and output :

- formatted output – printf =

int printf(char *format, arg 1, arg 2},...)

Ex:

```
void main() {          8
    printf(" %d", Printf ("%s", "ravindra"));
}
```

o/p : ravindra 8

Example:

```
/* Count number of set bits in x */.

int bitCount(unsigned x){
    int b;
for(b=0; x!=0; x>>=1)
        if (x & 1)
          b++;                          x = x>>1
    Return b;
    }
```

x = 1 1 0 0 0 0 0 1 ~~b is number of 1~~

(1) 0 0 0 0 0 0 0 1 b returns no. of 1.

b = ~~x~~ 3

- Formatted input-scanf:

✓int scanf (char, *format, ...)

"10 20 30" %d %d %d

✓int sscanf (char *string, char *format, args1, args2,)

```
{int day, month, year;

 scanf ("%d %d %d", &day, &month, &year);
```

- File Input Output:

- File Handling in C: <stdio.h>

```
     FILE *fp;
fp = FILE *fopen(char *name, char *mode)
     int fclose ( FILE *fp) .
```

fopen() → creat a new file (or) open existing file.
fclose() → Closes a file
getc() → reads a character to a file.
putc() → write a character to a file.
fscanf() → reads a set of data from a file.
fprintf() → writes a set of data to a file.
getw() → reads an integer from a file.
putw() → writes an integer to a file.
fseek() → set the position to desire point.
ftell() → gives current position in the file.
rewind() → set the position to the begining point.

example:

```
#include <stdio.h>
  void mind(){
  FILE *fp;
  int len;
  fp= fopen ("file.txt", "r");

  if (fp == NULL){
     printf("Error opening file");
  }
  fSeek(fp,0,SEEK-END);
   len = ftell (fp); → get file size by using it.
   fclose(fp);
   printf(" Total size of file.txt = %d byte\n",len);
   }.
```

(fp, -2, 2) → go left.

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

0 successful

Non-0 fail

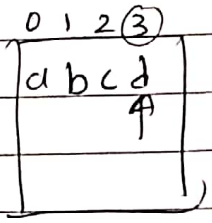
Whence

SEEK-SET	0	Begining of file.
SEEK-CUR	1	current position of file pointer.
SEEK-END	2	End of file.

(fp)

long int ftell(FILE *stream)

③ ↳ current position return.



void rewind(FILE *stream)

- puts(), gets()

(char *s)

char* gets(s) :- function reads a line from stdin (standard i/p) into the buffer pointed to by s until either a terminating newline (or) EOF.

(char *s)

int puts(s) :- function writes the string s and a trailing ~~w~~ newline to stdout.

```
#include <stdio.h>
 void main() {
   char str[100];
    printf("Enter a string \n");
    gets(str);
    puts(str);  }
```

O/p: R Nama
R Nama.

- Relationship between putc(), getc(), putchar(), getchar()

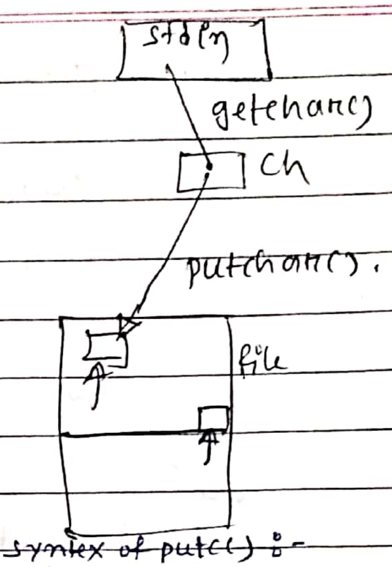
```
stdin
stdout
stderr
 └> linux environment
```

```
# include <stdio.h>
void main() {
    FILE *fp;
    char ch;
    fp= fopen("test.txt", "w");
    printf("Enter data");
                       or get(stdin)
    while ((ch= getchar())!= EOF){

        putc (ch, fp)
    }
    fclose (fp);

    fp= fopen("one.txt", "r");
    while ((ch=getc(fp))!=EOF) {
        printf ("%c", ch);
     or putc (ch, stdout);
    }
}
```

- file reading and writing by using putc() and getc()

```
# include <stdio.h>
void main() {
FILE *fp;
char ch;
fp = fopen ("~~text~~test.txt", "w");
printf(" Enter data");

while((ch = getchar())!=EOF) {
    putc (ch, fp);
}
```

if this file is not present then It will be newly created.



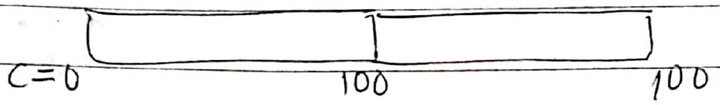
```
fclose(fp);
fp = fopen("one.text","r");

while ((ch=getc(fp))!=EOF){
    printf("%c",ch)
  }
   fclose(fp);
  }
```

syntex of putc() :-
int putc(int c, FILE *fp)
int getc(FILE *fp)

- W.A.P to read stream of characters :

```
#include <stdio.h>
#include <stdlib.h>
#define DEFAULTSIZE 100

 int ~~reg~~ resize(char *p, int count);

 void main(){
 int count=0, capacity= DEFAULTSIZE;
 ~~capali~~ char *input;
        char ch;
    input = (char *)malloc (DEFAULTSIZE);
    while ((ch = getchar())!= EOF){
        if (count == capacity){
          input= resize(input, capacity);
          capaity = capaity + DEFAULTSIZE;

          input[count++] =ch;
      }
          puts(input);
     }
```

```
Char *resize (char *p, int capacity) {

        return realloc (P, capacity + DEFAULTSIZE);
    }
```

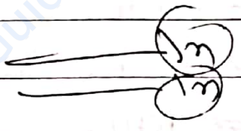


EOF -: Ctrl +d (in linux)
Ctrl +Z (in window)

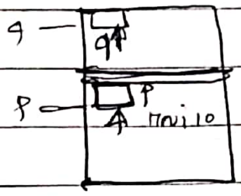
- Write a c-programm to count inputlines.

```
    # include<stdio.h>

        void main(){
         int lineCount, c;

         while ((c = getchar())) != EOF) {

            if (c == '\n')
              ++lineCount;
            }
        printf(" %d", lineCount);
            }
```

take i/p from user.



- W.A.P by using fscanf(), fprintf() =



```
# include <stdio.h>
  struct emp {
     Char name[10];
      int age;
     };
  void main() {
     struct emp e;
     FILE *P, *q;
     p= fopen("test.text", "a");
     q = fopen("test.text", "r");

     printf(" Enter name and age ");
     Scanf(" %s %d", ename, &e.age);

      fprintf(P, " %s  %d", e.name, e.age);
      fclose(P);

      do {
          fscanf (q, " %s  %d", e.name, e.age);
         printf(" %s %d", e.name, e.age);
         } while (!feof(q));

         }
```

append mode

feof(q) ─┬─ No zero ~~[illegible]~~ -(EOF)
         └─ 0 -(!EOF)

- **C Flow Control Statements:**

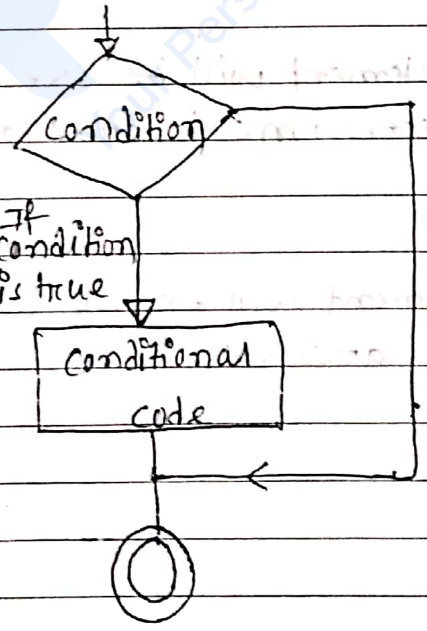
C provides two types of flow controls =

- → Branching (deciding what action to take)
- → Looping (deciding how many times to take a certain action)

- **Branching:**

① if statement:

```
(i) if (Boolean expression)
      {
          Statement        /* statement will be executed
                            if the boolean expression is
                            true */.
      }
```

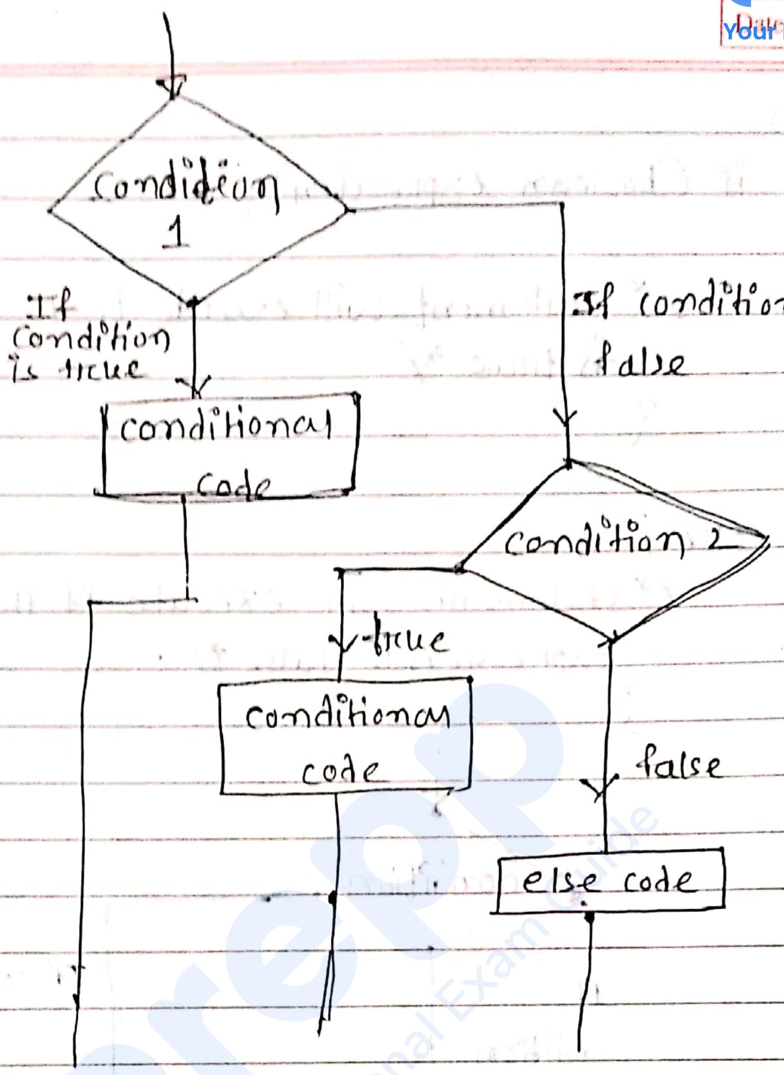


examples =

```
#Include <stdio.h>
int main() {
    int a=10;
    if (a<20) {
    printf("a is less than 20");
    }
    return 0;
}
```

(ii)

```
if (boolean expression1) {

    /* Statements will execute if boolean
       expression1 is true */
  }
else if (boolean expression2) {

    /* statement will be execute if boolean
       expression2 is true and 1 is false */
  }
else {
   /* Statement will execute when both expression
      1 & 2 are false */
  }
```

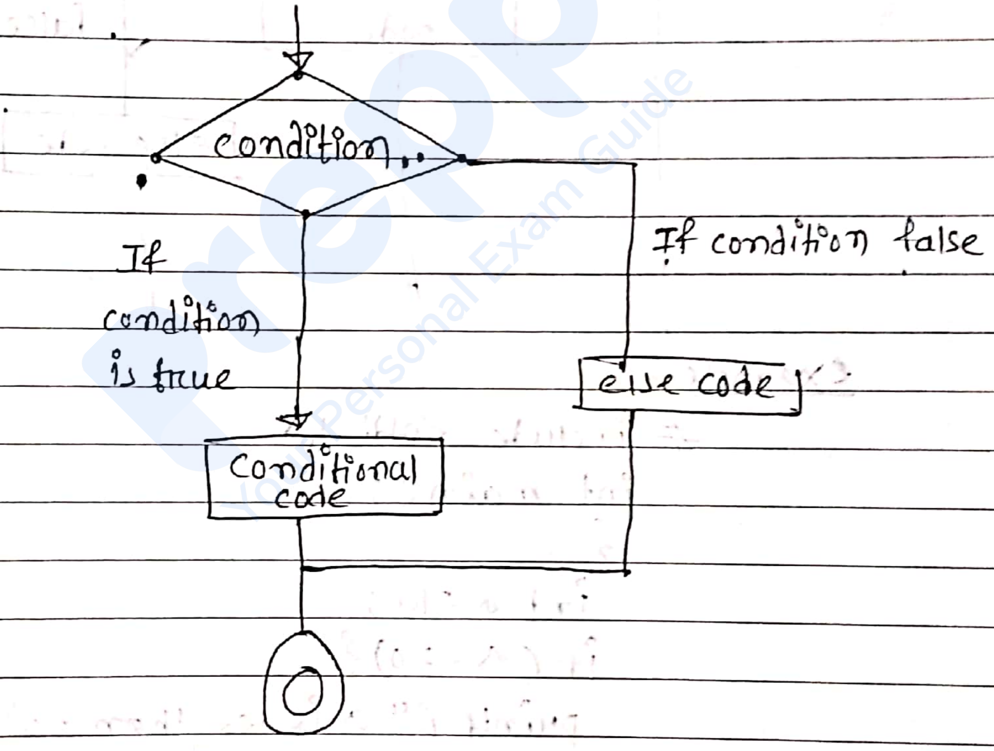


example :

```
# include <stdio.h>
    int main()
    {
      int a =10;
      if (a<20){
      printf ("a is less than 20"); }

      else if (a<100) {
       printf("a is between 20 and 100"); }

      else {
       printf ("a is greater than 100");
        }
       return 0;
      }
```

(iii)

```
if (boolean expression) {

    /* statement will execute if the boolean expression
       is true */
    }

else {
    (*statements will execute if the boolean to
      expression is false */.
    }
```



examples: ①

```
            # include <stdio.h>
              int main() {
               int a=10
               if (a<20)
               { printf("a is less than 20");
               else {
                 printf("a is greater than 20"); }
              return 0;  }
```

Page No.
Date :

example = ②

W.A.P to check wether a given number is even or odd.

```
→   # include <stdio.h>
    # Include <conio.h>
   void main () {

    int integer;
    printf("enter a integer:");
    scanf("%d", & integer);

    if ( integer %2 = = 0) {
       printf(" ~~ent~~ even number:");

   else
       printf(" odd number:");

   } getch();
   }
```

example = ③

W.A.P to check the largest number from given number.

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

     int main () {
      int a, b, c;
      clrscr();
      printf("Enter three numbers:");
      scanf("%d %d %d", &a, &b, &c);
```

Date :

```
if (a>b) {
  if (a>c)
    {
      printf("%d is the largest number",a);
    }
 }
else if (b>a) {
     if (b>c) 
     {printf(" %d is the largest number", b);
     }

else
 { printf(" %d is the largist number", c);
   }

~~gt~~ getch();
  return 0;
}
```

② Switch Statement :

```
switch(control variable)
 {
   case constant-1:  statement(s);
                     break;
   case constant-2 : statement(s);
                     break;
   :
   case constant-n : statement(s);
                     break;
  default : statement(s);
  }
```

example : ①

```
#include<stdio.h>
#include<conio.h>
 void main() {
 int week-day;
 printf("enter weekday");
 scanf("%d" & weekday);

switch (weekday) {

   case 0 : printf ("Monday"); break;
   case 1 : printf ("Tuesday"); break;
   case 2 : printf ("wednesday"); break;
   case 3 : printf ("Thrusday"); break;
   case 4 ; printf (" Friday"); break;
   case 5 : printf ("saturday); break;
   case 6 : printf ("sunday"); break;
 default : printf (" invalid");   }  }
```

Example-②

Write a program to make simple calculator.

→

```
#include <stdio.h>
# Include <conio.h>
void
~~int~~ main () {
  int operator;     /* char operation */
  double a, b;
 printf (" Enter an operation:" )   /* Enter (+,-,*,/)
 printf (" \n 1. addition. \n ~~sub~~ 2. subtraction.
            \n 3. Multiplication. \n 4. division.") .
scanf (" %d", & operator);

printf (" Enter two operands:");
scanf (" %lf %lf, &a, &b);

switch (operator) {

  case '+' : printf ("addition of a& b: %lf ", a+b);
  case '-' : printf ("sub of a&b: %lf", a-b); break; break;
  case '*': printf (" Multi of a&b: %lf", a*b); break;
  case '/': printf ("division of a&b: %lf", a/b); break;

 default : printf (" Invalid choice ") ;
   }

 getch();
}
```

③ Conditional Operators $(\overset{\text{if}}{?}\ \overset{\text{else}}{:}) =$

Syntax:

expression1 ? expression2 : expression 3

- expression 1 is condition.
- expression 2 is statement followed if condition is true.
- expression 3 is statement followed if condition is false.

~~example:~~

~~. x = 2~~

~~(x < 3) ? printf("true"); : printf("false");~~

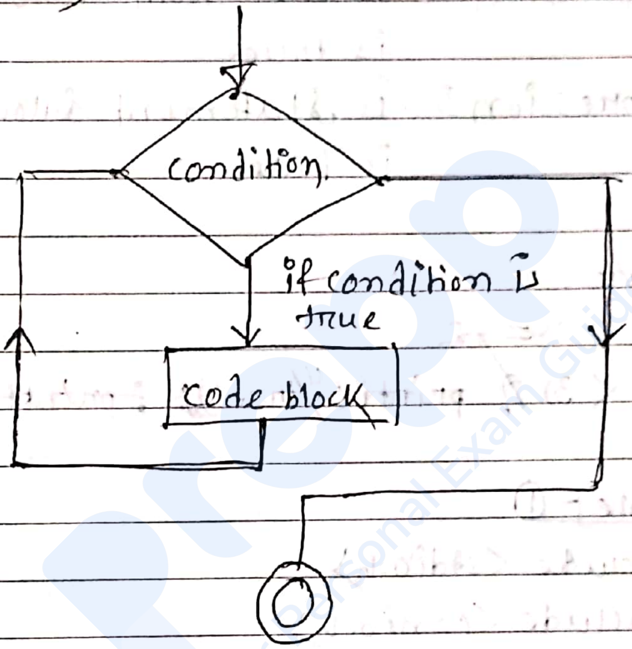
example ①

```
#include <stdio.h>
# include <conio.h>
    int main () {
      int age;
     pf(" Enter your age:\n");
     scanf(" %d", & age);

     (age >= 18) ? printf(" you are eligible to vote"):
                   printf("not eligible to vote");
     return 0;
}
```

• Loop Control Structure :

(i) While loop :

```
while (condition)
 {
   /* set of statements */
 }
```

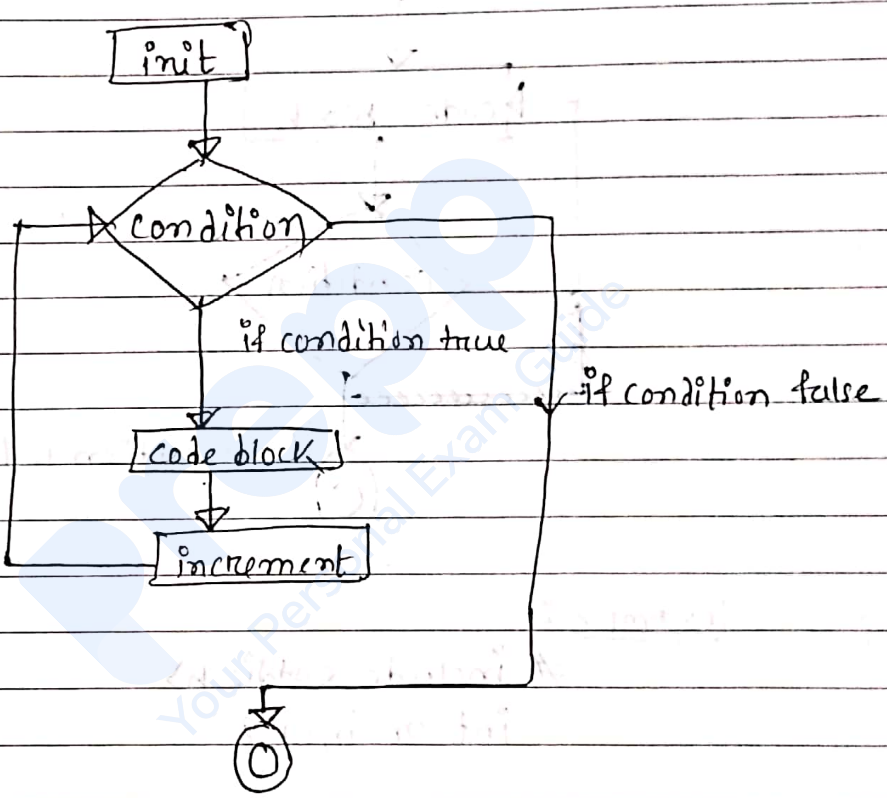


example = (stdio → standard i/o.)

```
# Include <stdio.h>
 void main()
  {
   int a =10;
   while (a<20) {
      printf("a value: %d", a);
      a++;
     }
  }.
```

(ii) for-loop :

```
for (initialisation ; condition ; increment/decrement)
  {
        conditional code ;
  }
```

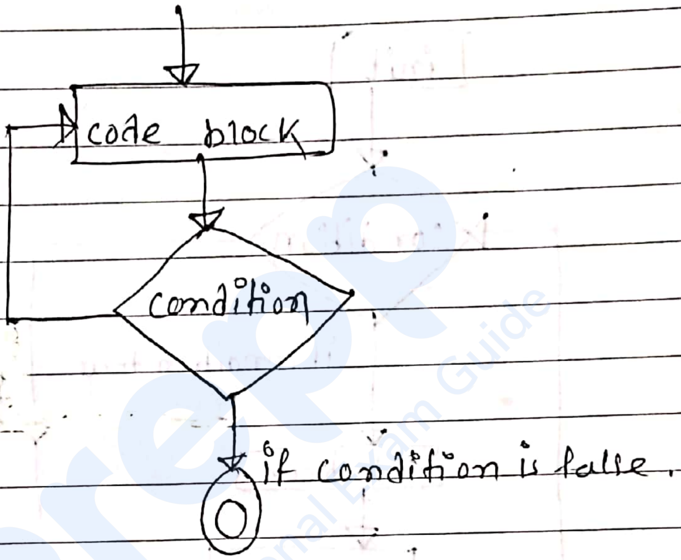


Example =

```
# include <stdio.h>
void main() {
    int a;

    /* for loop execution */
    for (a=0; a<20; a=a+1) {
        printf(" value of a: %d", a);
    }
}
```

(iii) Do-While loop :

```
do {

/* statements */
} while (Condition);
```



example :

```
# include <stdio.h>
  int main() {

     int a =20;
     do {
       print ("a value:%d", a);
       a ++
       } while (a<20);
  }
```

examples =

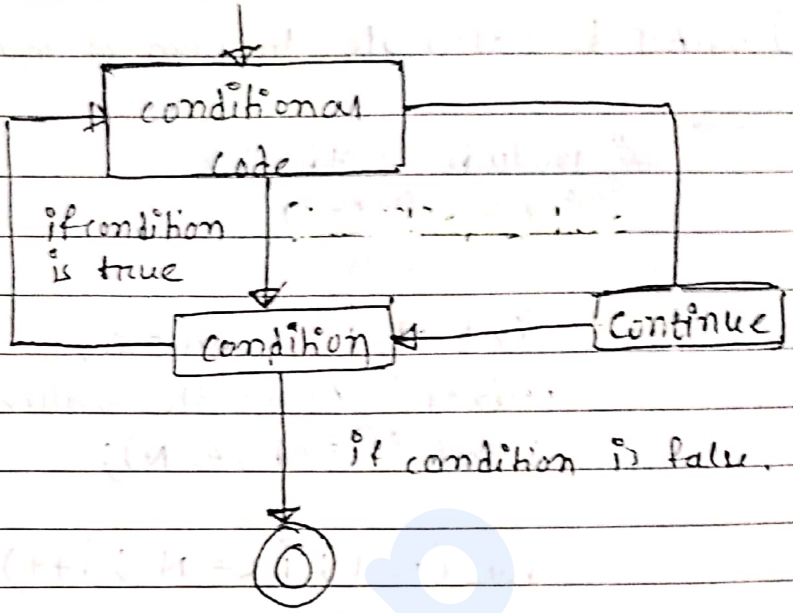
① W.A.P to calculate the sum of natural numbers.

→
```
# include <stdio.h>
  int main() {

     int N, i , sum = 0;
    printf("Enter the value of N:");
    Scanf ("%d", & N);

    for (i=1 ; i <= N ; i++) {

      sum = sum + i ;
    }
    printf(" sum of ~~%d~~ Natural number %d", sum);
    getch();
    return 0;
  }
```

② W.A.P to read input untill user enter a positive integer.

→
```
 # include <stdio.h>
   int main(){
     int n ;
   do {
     printf("Enter a value:");
     Scanf ("%d", n);
    } while (n <=0)
    print("n value : %d", n);
    return 0;
  }
```

- continue statement =



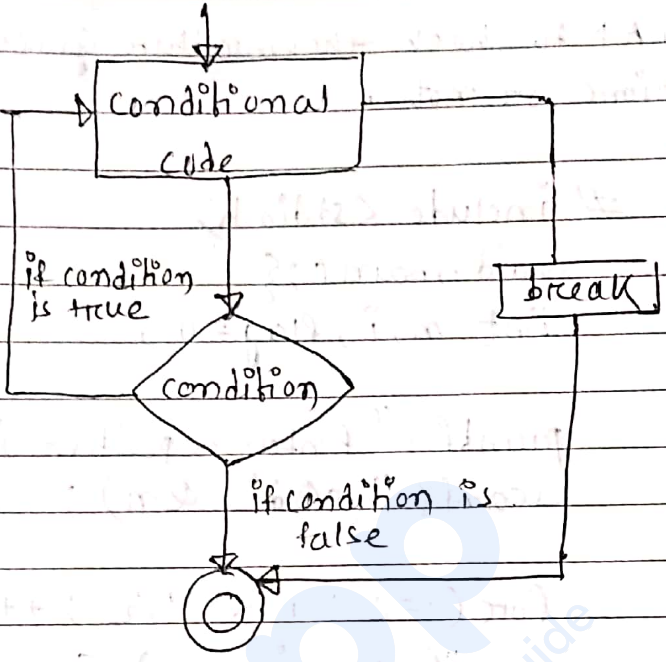
example :-

write a program to read 15 integers from user and print sum of ~~a~~ only positive integers.

→

```
# include <stdio.h>
void main() {
int i, n, sum = 0;
for (i=0; i<15 ; i++)
 {
   printf("Enter integer:");
   scanf("%d", &n);

     if (n<=0)
     continue;
   sum = sum + n;
  }
printf("sum of positive integers=%d", sum);
getch();
return 0;
}
```

- break statement :



Example :

W.A.p to read n integers until user enters a negative integer or number of integers read reaches to 15 .

→

```
# include<stdio.h>
  void main() {
     int n, count, i;
   for (i = 0 ; i<15 ; i++) {
     printf(" Read integer");
     scamf("%d", &n);
     if (n<0){
       break ;
     }
   }
}
```

Example

① W.A.P to check ~~the~~wheather given number is prime or not.

→

```
# include <stdio.h>
  void main(){
  int n,i, flag=0;

  printf ("Enter a positive integer:");
  scanf("%d", &n);

   for(i=2 ; i<=n/2 ; ++i) {
        if (n%i==0) {
            flag=1;
            break;
        }
   }
 if (flag ==0)
   printf(" %d is a prime number",n);

 else
   printf("%d is not a prime number",n);
  }
}
```

Example-2

W.A.P to find factorial of a given number.

→

```
# Include <stdio.h>
  void main () {                                  → for big data type.
      int n, i;
range (0-(2^64-1)) — Unsigned long long factorial = 1;

      printf(" Enter an integer : ");
      Scanf(" %d", &n);

      if (n<0)
         printf(" factorial of negative numbers does not
                                             exist");

      else {

         for (i=2; i<=n; i++){
            factorial = factorial * i; }

         printf ("factorial of %d = %llU", n, factorial);
         }                                  long long unsinged
                                                 data type
  }
```

Example-③

W.A.P to print half pyramid using *:

→
```
#include <stdio.h>
 void main() {
   ~~[illegible]~~;
   int i, j, numofrows;

 ~~for~~ printf("Enter the no.of rows :");
    scanf("%d", &numofrows);

 for (i=0 ; i<numofrows; i++) {

      for(j=0 ; j<=i ;j++) {

           print ("*");
        }

        print("\n");
    }

 }
```

numofrows = 5

i= 0, 1, 2, 3, 4

output:

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

Page No :
Date :

example : ④

W.A.P to count number of digits in an integer.

→

```
# include <stdio.h>

     void main() {
         int n, count = 0;
      printf(" Enter an integer:");
      Scanf ("%d", &n);
      while (n != 0) {

            n = n/10;
             ++ count ;
         }
       printf(" Number of digits: %d", count);

     }
```

output:

```
   Enter an Integer: 142
   Number of digits: 3.
```

$n = \frac{142}{10} = 14.2$ ①

$\Downarrow$ int

14

$n = \frac{14}{10} = 1.4$

$\Downarrow$

1

$n = \frac{1}{10} = 0.1$

$\Downarrow$

0

count = ~~0~~ ~~1~~ ~~2~~ ③

example : ⑤

W.A.P to Check Wheather given numbe is amstong amstrong or not.

→ amstrong number means,

$$371 = 3^3+7^3+1^3 = 371 \text{ (yes)}$$
$$121 = 1^3+2^3+1^3 = 10 \text{ (No)}$$
$$1648 = 1^4+6^4+4^4+8^4 = 1648 \text{ (yes)}.$$

→

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

void main(){
 int number, OriginalNumber, remainder, result=0,
                                        n=0;

 printf("Enter an integer");
 scanf("%d", &number);

  originalNumber = number;
  while (originalNumber != 0) {
      origiminalNumber/=10;
        ++n;
   }

   originalNumber = number;
   while (originalNumber != 0){
     remainder = originalNumber %10;
     result = result + pow(remaider, n);
   } originalNumber /= 10; }
 (result==number) ? printf("Armstrong Number"):
                 printf("Not Armstrong Number");

}
```

142
n=3

1 4 2

When i/p = 142
executes
$0+2^3$
2^3+4^3
$2^3+4^3+1^3$
$=8+64+1$
$=73$
output:
NO
armstrong
number

example - (6)

W.A.P to to pint the following pattern :

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

→
```
#include <stdio.h>
void main(){
int i, j, k, numofRows ~~##~~;
printf("Entern Number of Rows :");
scanf("%d", &numofRows);

                      (4)
for (i=1; i<=numofRows; i++){
             1          (4)
      for (j=i; j<numofRows; j++){
           printf(" ");
      }
                   1
        for (k=1; k<(i*2); k++){
              printf("*");
        }
        printf("\n");
     }
  }
```

working procedure =

```
(4-1) - - - *  - - -  (2*1-1)          i = 1✓ 2✓ 3✓ 4✓
(4-2) - - * * * - -   (2*2-1)
(4-3) - * * * * * -   (2*3-1)          - - - *
(4-4) * * * * * * *   (2*4-1)          - - * * *
                                       - * * * * *
                                       * * * * * * *
```

✓

example = (7)

W.A.P to check whether given number is palindrome or not.

→ like, -121
-11311
-2442

n=121
n%10

$R = 0 \times 10 + 1 = 1$
$= 1 \times 10 + 2 = 12$
$= 12 \times 10 + 1 = 121$

```
→ # include <stdio.n>
    void main() {
    int n, reversedNumber=0, remainder, original
                                        Number;
    printf("Enter a number :");
    scanf("%d", &n);
    Original number = n;
                                          12(1)
    while(n != 0){
(heart)   remainder = n % 10;
          reversedNumber = reversed Number * 10 +
                                        remainder;
          ~~rev~~ n /= 10;
    }
  (originalNumber == reversedNumber) ? printf("
              palindrome") : printf("not a
              palindrome");
    }
```

example-⑧

W.A.p to generate fibanacci sequences given first number and second no. of sequence.

→ 0, 1, 1, |2, 3, 5, 8, 13, . . .
f s sum

```
#include <stdio.h>
 void main() {
  int first, second, sum, num, counter=0;
                              (4)

   printf("Enter the number of terms:");
   scanf("%d", &num);
   printf("Enteter first number:");
   scanf("%d", &first);
   printf("Enter second number:");
   scanf("%d", &second);

  printf("Fibonacci series %d %d", first, second);

   while (counter <num) {
                    (4)
heart.   sum = first+second;
          printf("%d", sum);
          p,
          first = second ; second =sum;
          counter++;
       }
 }
```

working = num=3 counter=0,1,2 | first=2 second =3

2 | 3 ⑤ ⑧ ⑬
f s

Page No.
Date

- Functions :

Syntax of function Defination :

```
                                              (argument)
return-data-type function-name (data-type var1, data-type
                                             var2 ...)
    {
     /* function-body */
    }
```

- Return type :-

A function may return a value. some functions may perform the desired operations without returning a value. In this case, the return-type is the keyword void.

example: Multiplication of two number using function.

```
#include <stdio.h>
#include<conio.h>
 int Multiplication(int, int);

int main()  → function name
{ int i, j, k ;
  Clrscr();
  pf(" Enter two value:");
  sf("%d %d", &i, &j);
  k = multi (i, j);
                 → actual parameters.
  pf("%d \n", k);
  return 0; }
```

function body ←

formal parameters.

fun-ction name ← int Multi (int x, int y) {

```
int Multi (int x, int y) {
  int a; a = x * y; return a; }
```