

C Programming Language

GATE CSE NOTES

C-LANGUAGE

classmate

Date _____

Page _____

• Intro programming in C :

- (1) All C programs must 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 modifiers with their Range:

Types of Modifier	Size(in byte)	Range of values
int	2	-2^{16-1} to $+2^{16-1}$
signed int	2	-2^{16-1} to $+2^{16-1}$
unsigned int	2	0 to $(2^{16}-1)$
short int	2	-2^{16-1} to $(2^{16}-1)$
long int	4	-2^{32-1} to $(2^{32}-1)$
float	4	$-2^{23}(3.4E+18)$ to $(3.4E+99)$
double	8	$-(1.0E+308)$ to $(1.0E+308)$
char	1	-2^{8-1} to (2^{8-1})
unsigned char	1	0 to (2^8-1)

• Types of operators:

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

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

(3) Relational operators (del <, <=, >, >=, !=, ==)

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

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

(6) Special operators (sizeof(), Ternary operator)

(7) Pointer operators (* - value at address, & - Address 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 a; 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 Ivalue:

→ expression that refers to a memory location are called "Ivalue" expression.

→ An Ivalue may appear as either the left hand or right hand side of an assignment:

$$\begin{array}{l} \text{Ivalue} \\ b = 10 \end{array}$$

$$a = b$$

- 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=5; }

int c; }

→ Local variable.

c=a+x;

printf(" Sum = %d", c)

}

output: 5+10 = 15

• Operators in C :

operators precedence

() [] → .

! , ~ , ++ , -- , + , - , * , & , (type) , size-of
A) { + , / , %

S) << , >>

C) { < , <= , > , >= ,
 == , !=

B) { &

I) { ^

L) { ::

D) { == , += , -= , *= , /= , % = , &= , ^= ,
 != , <<= , >>=

E) ,

Associativity

Left to Right

Right to Left

Left to Right :

)

)

)

)

)

)

)

)

)

Right to Left

Right to Left

~~Left to Right.~~

Left to Right

(to remember)

• Format Specifiers :

10 %10d → prints as decimal number.

----- %10.6d → prints as decimal number, at least 6 characters wide.

%10f → prints as floating point.

%10.6f → prints as floating point, at least 6 characters wide.

--- %10.2f → prints as floating point, 2 characters after decimal point.

----- %10.6.2f → print as floating point, at least 6 wide and 2 after decimal point.

%10c → print as ascii character.

%10lf → format specifiers for double.

- Character Input and output :

`getchar()` :- it reads the next input character from a text stream and returns that as its value.

`c = getchar()`

the variable `c` contains the next character of input.

`putchar()` :- `putchar` prints a character each time it is called.

example :-

`/* copy input to output */`

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

Linux commands

- ① `vi filename.c`
- windo
- :wq! → exit
- ② `gcc filename.c`
- (compile this file)
- ③ `./a.out`
- (to run)

\$./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	Scope (Visibility)	Lifetime (Alive)
auto	Memory	garbage	Inside a function or Block	Within the function/block	Until the function/block complete
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 terminates
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) int main() {

```
X register int i=10;
int *a=&i;
printf("%d", *a);
return 0;
}
```

O/P: error.

i value stored in register.
 but register \neq have no
 address. so compiler error
 can be occur in this case.

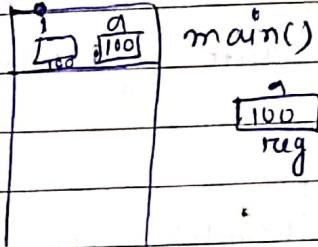
(2) int main() {

```
✓ int i=10;
register int *a=&i;
printf("%d", *a);
return 0;
```

}

O/P: 10.

activation record.



(3) int main() {

X int i=10;

```
register static int i=10;
printf("%d", i);
return 0;
```

}

O/P: compiler error

\rightarrow storing the value of i
 and in two ^{diff} places ~~also~~
~~so it not possible.~~

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", CountFunctionCall());

return 0;

}

O/P : 1

1 times function is called.

Stack section of process	
main()	
Count	CountFC()
1	
Count	CountFC()
1	
Count	CountFC()
1	
1	Pfc()
Count	CFC()
1	

④ ② int countFunctionCall (void) {

```
static int count;
return ++count; }
```

(Data section) of process

count	1
	2 3 4

int main () {

CountFunctionCall();

CountFunctionCall();

CountFunctionCall();

printf ("%d times function is called", CountFunctionCall());

return 0;

}

O/P : 9

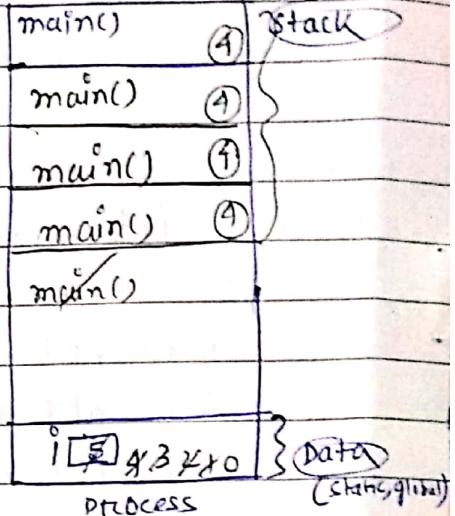
4 times function is called.

(5) What is the output of the following program?

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

```
int main() {  
    1 static int i=5;  
    2 if (--i) { 3 main();  
        4 printf(" %d", i);  
        3  
    }  
}
```

O/P: 0 0 0 0



(6)

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

```
int i;  
void fun1()  
{  
    i=20;  
    printf(" %d", i);  
}
```

```
void fun2()  
{
```

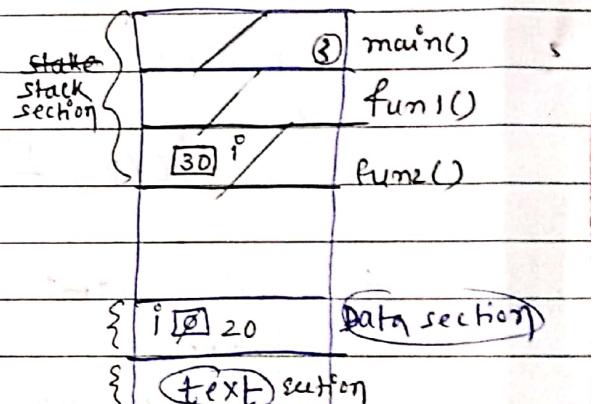
```
    int i=30;  
    printf(" %d", i);  
}
```

```
int main()  
{
```

```
    1 fun1();
```

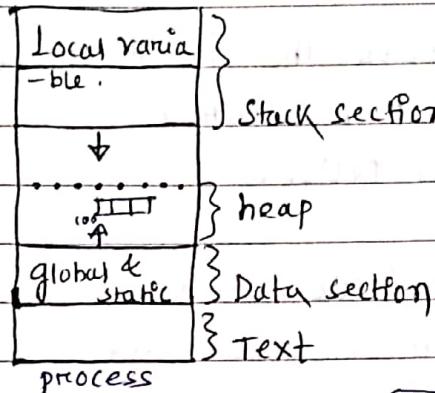
```
    2 fun2();
```

```
    3 return 0;  
}
```



O/P: 20 30

Storage Management: #include <stdlib.h>



① void *malloc (size-t n) → how many elements want to store atleast
Unsigned data-type size ^ 16 bits.

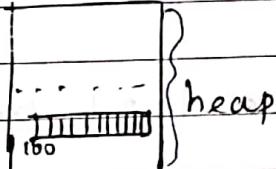
~~code example~~
ex: void *malloc (000) (sizeof (10))

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

int *i;

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

i++;



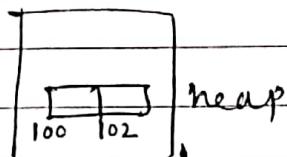
② void *calloc (size-t n, size-t size)

↓ how many element

you want to store

what is the size of each element

→ malloc and calloc always gives space in contiguous manner.



③ `void *realloc(void *ptr, size_t size)`

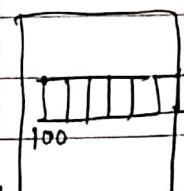
→ it used to increase the size of space.

→ if space is not available
then it returns NULL.

~~if space~~

Ex: `void *realloc(100, sizeof(20))`

→ pointer of allocated space and increase upto 20 byt.



④ `void free(void *ptr)`

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

→ to avoid memory leak problem we use free.

- Input and output :

- formatted output - printf =

`int printf(*char *format, arg1, arg2, ...)`

Ex:

```
void main() {  
    printf("%d", printf("0/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++;
    return b;
}
```

$x = 1100000(1)$

$\underline{1} \underline{1} \underline{0} \underline{0} \underline{0} \underline{0} \underline{0} \underline{0} \underline{0}(1)$

bit number of $\#1$

b returns no. of 1 .

$$b = x \# 3$$

- Formatted input - scanf:

int scanf (char *format, ...)

int sscanf (char *string, char *format, ^{"10 20 30"} _{and so on}, 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()` → creates a new file (or) open existing file.

`fclose()` → closes a file

`getc()` → reads a character to a file.

`putc()` → writes 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 main() {
```

```
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);
```

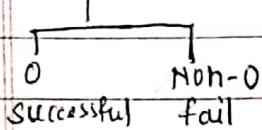
```
printf ("Total size of file.txt = %d bytes", len);
```

```
}
```

go left

(fp, -2, 2)

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



Whence ←

{ SEEK_SET 0 Beginning of file.

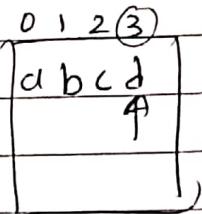
SEEK_CUR 1 current position of file pointer.

SEEK_END 2 End of file.

(fp)

long int ftell(FILE *stream)

③ ↳ current position return.



void rewind(FILE *stream)

- puts(), gets()

char *gets(^{char **s})

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

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

#include <stdio.h>

void main()

char str[100];

printf("Enter a string\n")

gets(str);

puts(str); }

O/P: R Name

R Name.

- 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");

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, ~~fp~~ stdout);

}

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

#include <stdio.h>

void main() {

FILE *fp;

char ch;

fp=fopen("texttest.txt", "w");

printf("Enter data");

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

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

 putc(ch, fp);

}

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

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

{

```
fclose(fp);
```

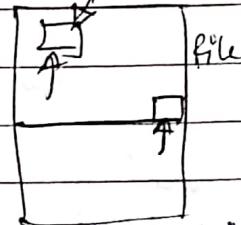
{

std::in

getchar()

ch

putchar().



syntax 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 reresize(char *p, int count);
```

```
void main() {
    int count = 0, capacity = DEFAULTSIZE;
    char *input;
    char ch;
    input = (char *) malloc(DEFAULTSIZE);
    while ((ch = getchar()) != EOF) {
        if (count == capacity) {
            input = resize(input, capacity);
            capacity = capacity + DEFAULTSIZE;
        }
        input[count++] = ch;
    }
}
```

```
input[count] = ch;
```

{

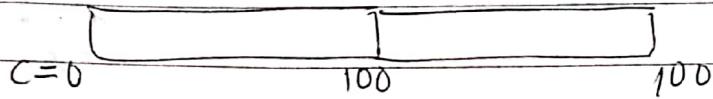
```
puts(input);
```

{

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

return realloc(p, capacity + DEFAULTSIZE);

}



EOF - Ctrl + d (in Linux)
Ctrl + Z (in window)

- Write a c-program to count inputlines.

#include<stdio.h>

void main() {

int lineCount, c;

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

if (c == '\n')

++lineCount;

}

printf(" O/p ", lineCount);

}

}

• 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", e.name, &e.age);
```

```
fprintf(p, "%s %d", e.name, e.age);
```

```
fclose(p);
```

```
do {
```

```
fscanf(q, "%s %d", e.name, &e.age);
```

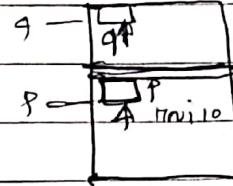
```
printf("%s %d", e.name, e.age);
```

```
} while (!feof(q));
```

```
}
```

$\neg \text{No zero } (\text{content}) - (\text{EOF})$

$\neg 0 - (!\text{EOF})$



– time spent – time spent



- 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)

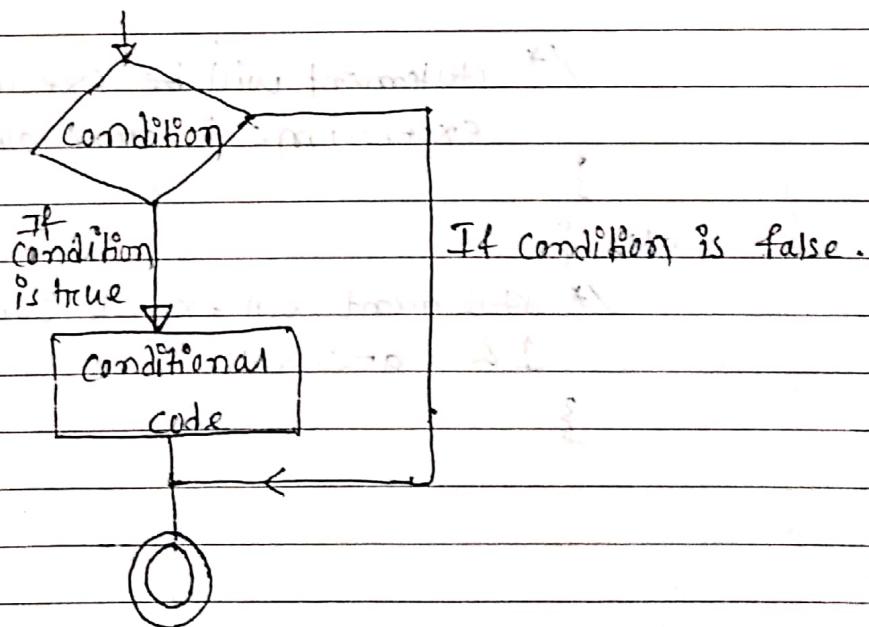
- Branching :

- ① if statements

(i) if (boolean expression); (execute code)

(ii) 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 expression-1) {
```

```
/* statements will execute if boolean  
expression 1 is true */
```

```
}
```

```
else if (boolean expression 2) {
```

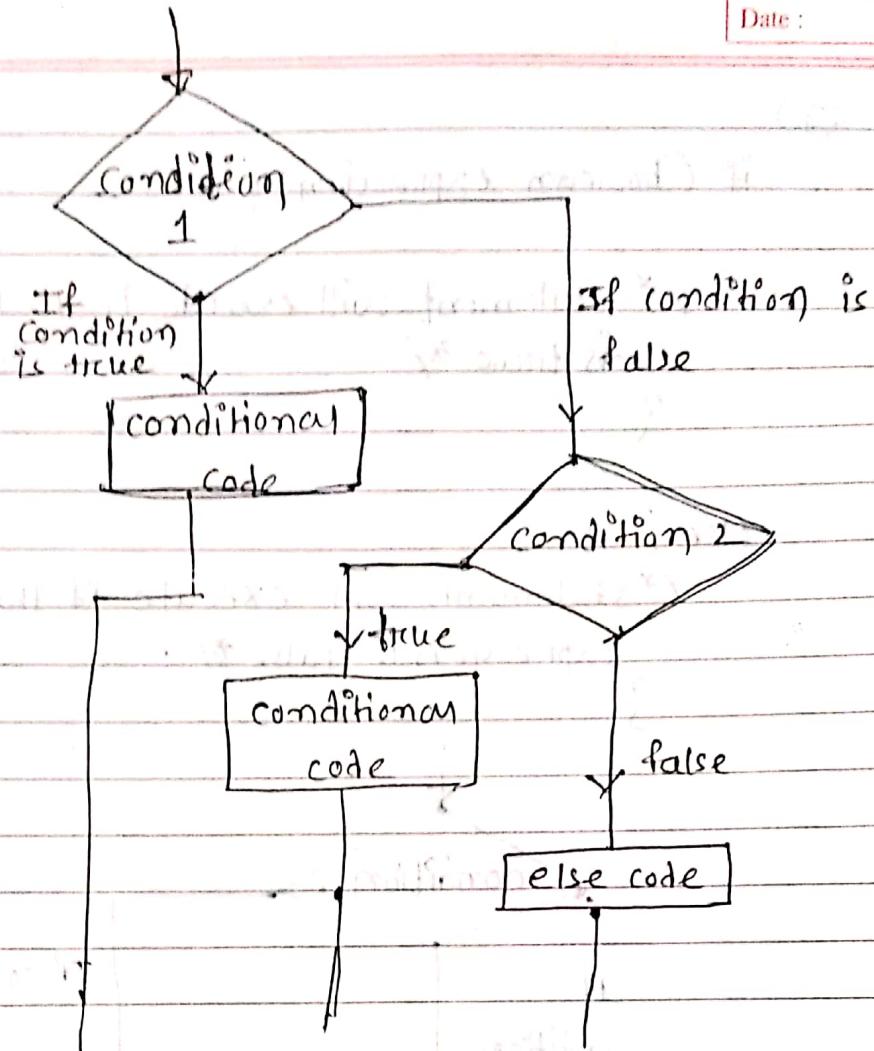
```
/* statement will be execute if boolean  
expression 2 is true and 1 is false */
```

```
}
```

```
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) {

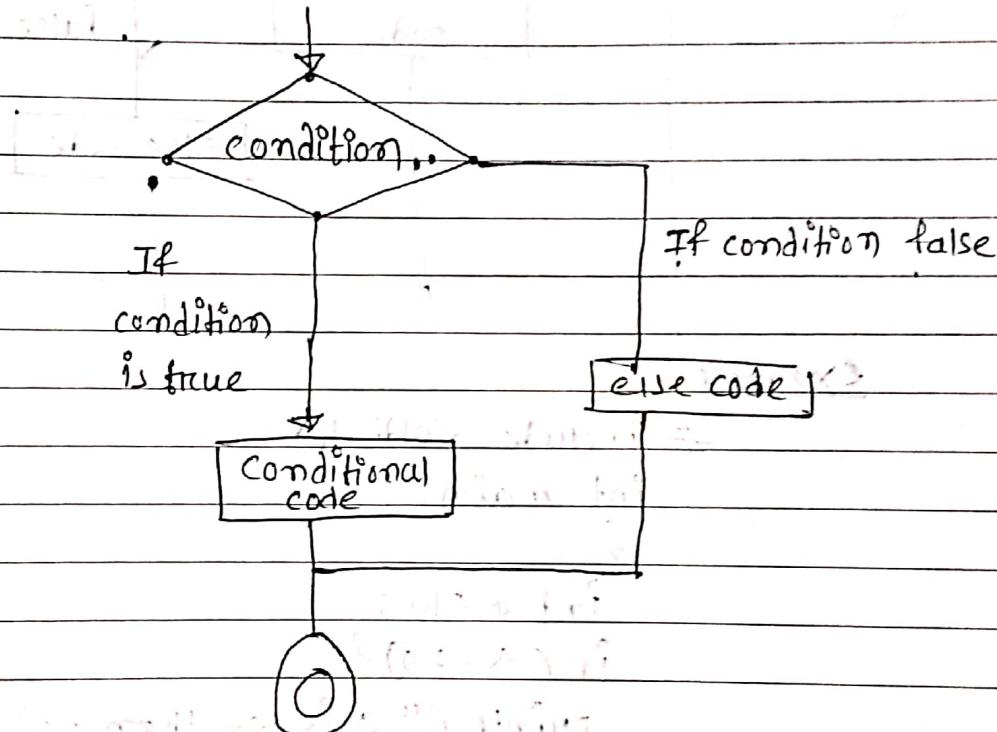
/* statement will execute if the boolean expression
is true */

}

else {

/* statements will execute if the boolean
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; }

Example = ②

w.A.p to check whether 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("Even number.");
```

```
    else
        printf("Odd number.");
```

```
    getch();
}
```

Example = ③

w.A.p to check the largest number from given numbers.

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

```
int main() {
    int a, b, c;
    clrscr();
```

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

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

```
printf(" enter weakday");
```

```
scanf("%d" &weakday);
```

```
switch (weakday) {
```

```
case 0 : printf ("Monday"); break;
```

```
case 1 : printf ("Tuesday"); break;
```

```
case 2 : printf ("Wednesday"); break;
```

```
case 3 : printf ("Thursday"); 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 operation; /* char operation */
```

```
    double a, b;
```

```
    printf ("Enter an operation: "); /* Enter A-Z */

```

```
    printf ("In 1. addition. In 2. subtraction.
```

```
In 3. Multiplication. In 4. division.").
```

```
    scanf ("%d", &operation);
```

```
    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;
```

```
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();
```

```
return 0;
```

```
/* Author: Shubham Kulkarni */
```

```
/* Date: 10/09/2018 */
```

```
/* File: SimpleCalculator */
```

```
/* Language: C */
```

(3) Conditional Operators (`? :`)

Syntax:

`expression1 ? expression2 : expression3`

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

Example:

```
// n = 2
(n < 3) ? printf("true") : printf("false");
```

Example - ①

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

```
int main() {
    int age;
    pf("Enter your age: ");
    scanf("%d", &age);
```

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

}

• Loop Control Structure:

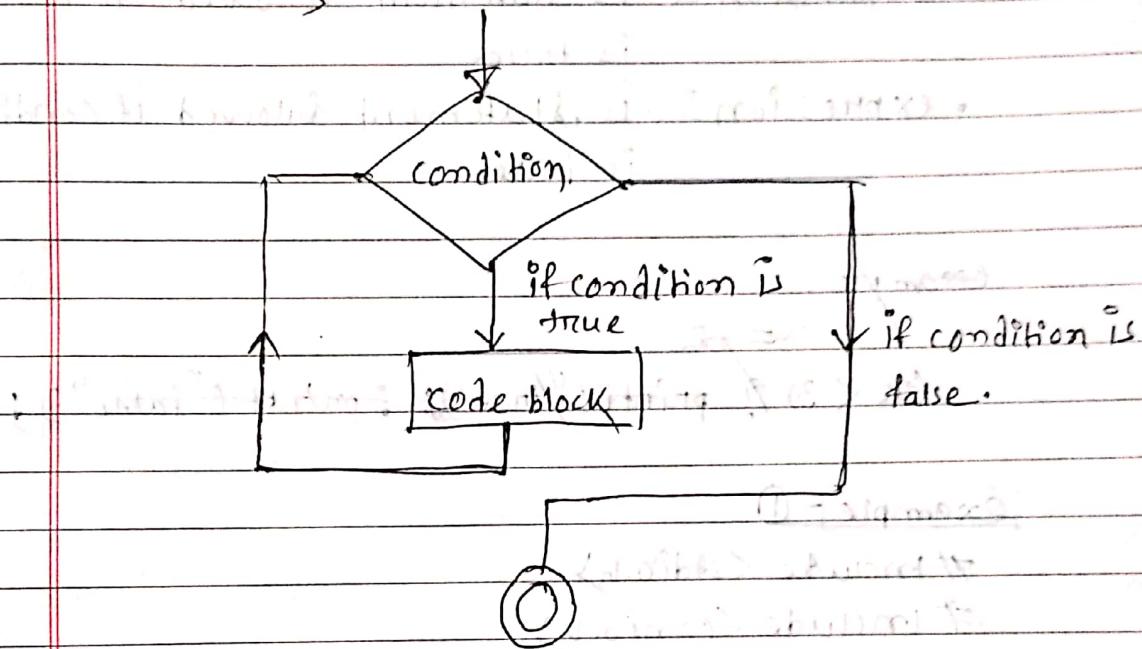
(i) While Loop:

while (condition)

{

 /* set of statements */

}



example =

for (; ;) (stdio → standard I/O.)

#include <stdio.h>

void main()

{

 int a=10;

 while (a<20) {

 printf("a value: %d", a);

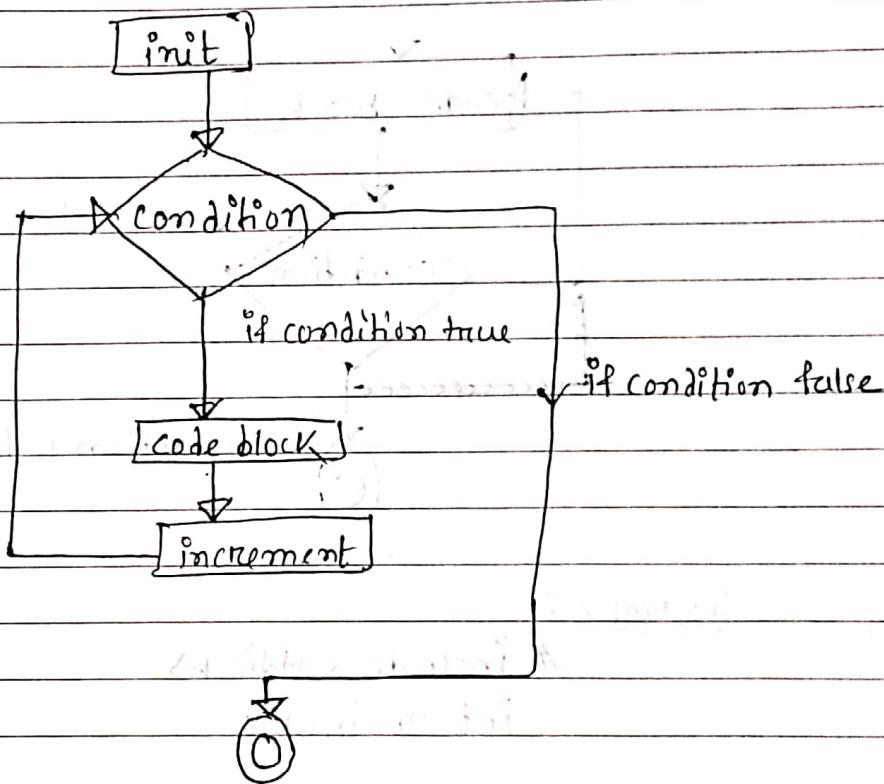
 a++;

}

}.

(ii) for - loop :

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



Example, =

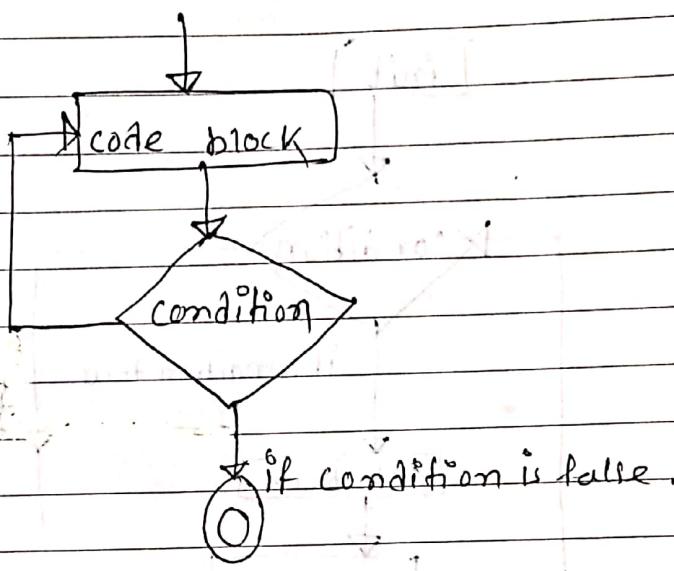
```
# include <stdio.h>
void main() {
    int a;

    /* 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 {

printf ("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 Natural numbers %d", sum);

getch();

return 0;

}

② W.A.P to read input until user enters a positive integer.

→

#include <stdio.h>

int main() {

int n;

do {

printf("Enter a value:");

scanf("%d", &n);

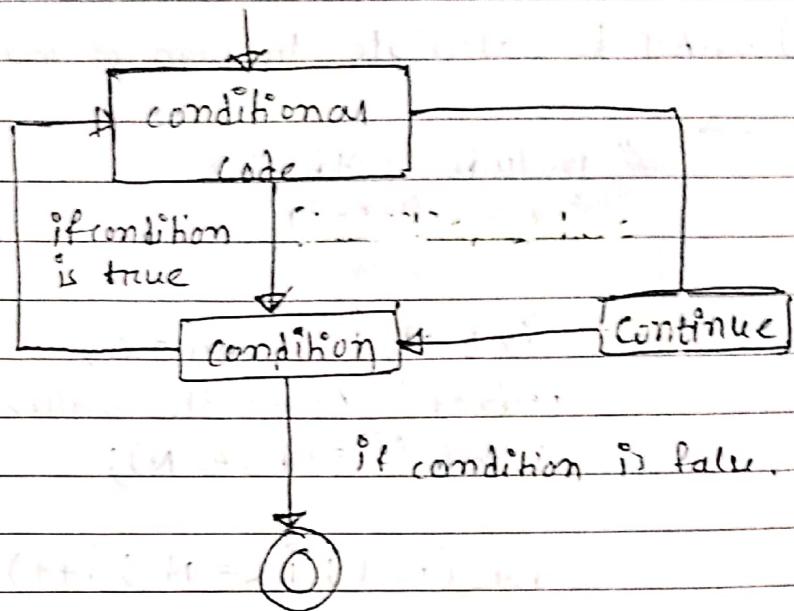
} while (n <= 0)

printf("n value %d", n);

return 0;

}

- Continue statement =



Example :-

write a program to read 15 integers from user and print sum of all 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;
    }
}
```

continue;

sum = sum + n;

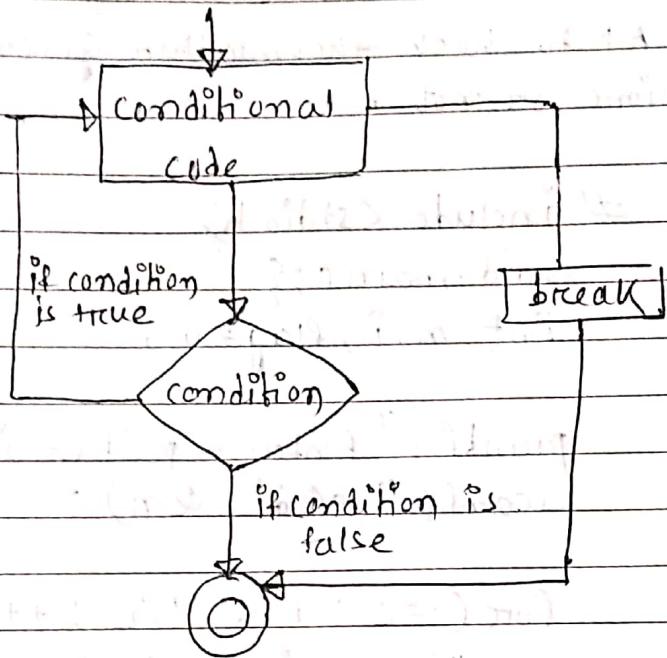
printf("sum of positive integers=%d", sum);

getch();

return 0;

}

- break statement:



(Example):

W.A.P to read 15 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.");
        scanf("%d", &n);
        if(n < 0)
            break;
    }
}
  
```

(Example)

- ① WAP to check whether 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);
```

```
}
```

```
8
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

```
\
```

```
/
```

</

Example-②

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



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

```
void main() {
```

```
int n, i;
```

~~range (0 - 2^{63})~~ — Unsigned long long factorial = 1;

→ For big data type.

```
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 unsigned
data type

```
}
```

ANSWER

Example - ③

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

```

→ #include <stdio.h>
void main() {
    int i, j, numofrows;
    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:

```

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

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.

$$n = \frac{142}{10} = 14.2 \quad \text{①}$$

\downarrow int

$$n = \frac{14}{10} = 1.4 \quad \text{②}$$

\downarrow

$$n = \frac{1}{10} = 0.1 \quad \text{③}$$

\downarrow

count = ~~0~~ × ~~3~~ ③

0

example ⑤

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



Armstrong 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 <math.h>

void main() {

int number, originalNumber, remainder, result=0,
n=0;

printf("Enter an integer");

scanf("%d", &number);

originalNumber = number;

while (originalNumber != 0) {

originalNumber /= 10;

+ n;

}

142

n = 3

142

originalNumber = number;

while (originalNumber != 0) {

remainder = originalNumber % 10;

result = result + pow(remainder, n);

originalNumber /= 10; }

(Result == number) ? printf("Armstrong Number") :

printf("Not Armstrong Number");

When I/P = 142

outputs

0+2

2^3+1^3

2^3+4^3+1^3

= 8+64+1

= 73

output:

No

armstrong

number

}

Example - 6

W.A.P to print the following pattern :

```

    *
   ** *
  *** *
 **** *

```

→ #include <stdio.h>

```

void main()
{
    int i, j, k, numofRows;
    printf("Enter Number of Rows : ");
    scanf("%d", &numofRows);
}
```

```

for (i=1; i <= numofRows; i++) {
    (4)
}
```

```

    for (j=i; j < numofRows; j++) {
        (4)
        printf(" ");
    }
    (4)

```

```

    for (k=1; k < (i*2); k++) {
        (4)
        printf("*");
    }
    (4)

```

```

    printf("\n");
}
(4)

```

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.h>

void main() {

int n, reversedNumber = 0, remainder, original
number;

printf("Enter a number:");

scanf("%d", &n);

Original number = n;

121

while (n != 0) {

(heart)

 remainder = n % 10;

 reversedNumber = reversedNumber * 10 +

 remainder;

 n = n / 10;

(originalNumber == reversedNumber) ? printf("palindrome") : printf("not a palindrome");

3

example - 8)

W.A.P to generate Fibonacci sequences given first number and second no. of sequence.

$\rightarrow 0, 1, 1, [2, 3, 5, 8, 13, \dots]$

f s sum

#include <stdio.h>

void main()

int first, second, sum, num, counter = 0;

printf("Enter the number of terms:");

scanf("%d", &num);

printf("Enter first number:");

scanf("%d", &first);

printf("Enter second number:");

scanf("%d", &second);

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

while (counter < num) {

 sum = first + second;

 printf("%d", sum);

 first = second; second = sum;

 counter++;

}

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

2 | 3 5 8 13
 f s

Widely used in the field of business.

Businessman can make his business more successful.

Businessman can increase his sales.

Businessman can increase his profit.

Businessman can increase his market share.

Businessman can increase his customer base.

Businessman can increase his brand value.

Businessman can increase his efficiency.

Businessman can increase his productivity.

Businessman can increase his revenue.

Businessman can increase his profit margin.

Businessman can increase his market share.

Businessman can increase his customer base.

Businessman can increase his brand value.

Businessman can increase his efficiency.

Businessman can increase his productivity.

Businessman can increase his revenue.

Businessman can increase his profit margin.

- Functions :

Syntax of function definition:

(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 numbers using function.

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

int main() → function name

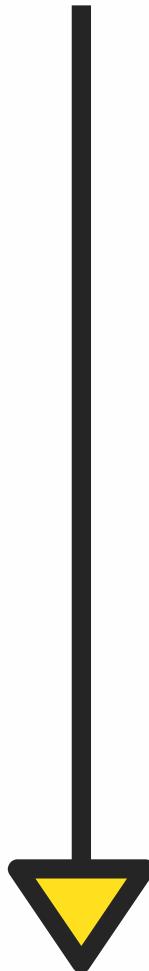
{ int i, j, k;
 clrscr();
 pf(" Enter two values: ");
 sf("%d %d", &i, &j);
 K = multi(i, j);
 pf(" %d \n", k);
 return 0; }

function body ←
 ↗ actual parameters.

fun- ← int Multi(int x, int y) {
 ↗ formal parameters.
 int a; a = x * y; return a; }

TO DOWNLOAD THE COMPLETE PDF

**CLICK ON THE LINK
GIVEN BELOW**



WWW.GATENOES.IN

GATE CSE NOTES