Module (1)

Introduction to C

gcc → gnu compiler collection

V(1) → Write a program to print Hello World!

```
# include <stdio.h>   # header file or library
                      also "stdio.h"
main(){
    printf("hello world")
}
```

printf → function
"hello world" → String constant

Every C program must contain main() function atleast once. C compiler are pre-understood to start the execution from main().

Each C program contains → function, Variables

function → Contains the command, what work to be done

Variables → store the values, useful while executing the programme

Step-1 Create a file hello.c with above progsemon
Step-2 Run gcc hello.c on terminal
Step-3 a.out

o/p → hello world.

V(2) - Introduction to C Language

Linux environment used in product based companies.

$ vi file.c vi → compiler name

open a terminal

save as :wq

then,

$ gcc file.c → compiler

$ ./a.out → Run the file

o/p hello world

$ touch <filename>
$ gcc <filename>
$ ./a.out
⇒ <filename.c>

① stdio → standard I/p O/p contains standard fn.

② # include { → pre-processor directive → Tell pre-processor to include content of
define {
<file>
↳ used for macro-expansion

Punch card → Machine lang
↓
Assembly lang (ADD)
↓
High level
↑
Low level lang

③ function() {
<function body>
}

function(x,y,z) : x, y, z are arguments

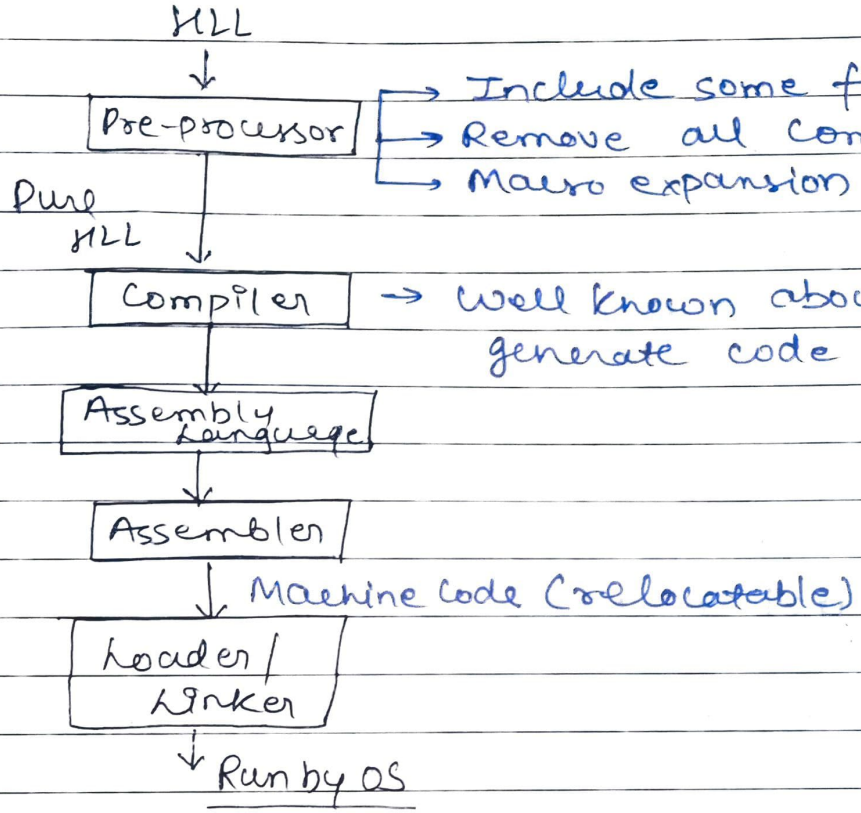
Argument sent to function, to use for execution.

compiler collection converts high level language into low level machine language (binary code)

C → High level language

Date :
Page No.

Phase involved in compilation



file.c
↓
gcc -E file.c [How file is look like after pre-processing phase]
↓ file.i
gss -S file.i [After compilation]
↓ file.s
gcc -c file.s
↓ file.o
gcc -o file.o [Used for link all files]
↓ a.out

final o/p will be store in 'a.out'

gcc -Wall -save-temps file.c → Create and save all the temporary files.

Date :
Page No.

V(3) format specifiers

%d → Print as decimal number
%6d → Print as decimal, at least 6 character wide
1 0
_ _ _ _ _ _

%f → Print as floating point
%6f → Print as float, at least 6 character wide
%0.2f → At least two character after decimal

%6.2f → At least 6 character wide, two after decimal
%c → Print as ascii character

Lots of other format specifier are also exist in C

V(4) Character Input and output

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

c = getchar()

the variable c contains the next character of input.

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

To check working

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

Read or write only a single character at a time.

Not Imp.

$ ./a.out < infile > outfile

① long long double → not exist

② unsigned long double ↓ unsigned cannot be used with double,

Types, Operators & Expressions

V(1) Data types and sizes

Char → single byte (8 bit) (0-255)
int → an integer (size Machine depend)
(min 32 bits) float → single precesion
(min 64 bits) double → double-precesion
long double → > double prevson

Quantifiers

Short } mainly use
long } with int as
i) short int i / short i
ii) long int i
iii) long i

Signed } int, character
Unsigned }

Ex- Unsigned int → If int of size 8 bit that it will hold only +ve numbers (0-255)

Signed int → (-128 - 127) in case of 8 bit into

Note:-

Char is generally unsigned only.

If number is greater than double then we can use long double.

<limits.h> or <float.h> and sizeof() use to get the size of datatype.

Variable

→ start with character
→ In between you can use number, alphabet or underscore (_)
→ start with _ also.

sizeof(int) → contains the size of int

You can find size of any data type using sizeof() and print it.

User defined data types

Array, pointer, Structure, Union, Enum.

Date :
Page No.

V(2) Enum Data Type (User defined data type)

enum used to define some enumeration constant.
It is a keyword.

Ex enum months { Jan, Feb, March, Apr, May, June };

Case-1 If no value given to any argument then Jan will automatically take 0 and each other month will take value with increment of 1 as
Feb = 1, Mar = 2 .... June = 5

Case-2 If Jan = 2 then,
Feb = 3, Mar = 4 ... June = 6.
Value will set with the increment of 1

Case-3 If Jan has no value but Apr = 4
then Jan = 0, feb = 1, Mar = 2, May = 5,
June = 6.

Case-4 If April = 1 then value assigned to Jan will be ?
Jan = 0, Feb = 1, March = 2 because no value assign to Jan.

Other than this we can use # define as-

```
# define NO  0
# define Yes 1
```

enum → It is a separate data type called enumerate.

Type Conversions: (i) In operations

Let, a + b then, it is better to convert
↓ ↘(32 bit) 16 bit no. into 32 bit, bcoz
(16 bit) it diminished the change of losing information. while in vice-versa case we will definitely loose some information

Step 1 → Detect the operand of highest size data type

Hierarchy

long double
double
float
short / int
char

(Size) ↑

Ex- char c
int i

i = c → no loss info.
c = i → losing info.

(ii) On assignment

Ex a = b
(16 bits) ↓ ↘(32 bits)

Convert RHS number to size of LHS.

Example

There may be loss of information some times.

Explicit Casting

Ex ① f = (float) 10 o/p f = 10.00
② i = (int) (24.4) o/p i = 24

↳ 'casting operator'

Constant : 1234 - int
1234l - long int
12.34 - float
0x37 - octal
0x12 - long hexadecimal
0x12l → long octal
'x' → character constant

① Const float Pi = 3.14
↓
Qualifier
If any one try to change it, it will show a error.

② # define Pi 3.14 ⇒ fast acces then first ① one.

String constants : Represent using double quote "string".

string 6 characters are present but it will take total 7 byte
1 byte extra for null '\0'.

/* strlen : return the length of string

```
int strlen (char s[]){
    int i=0
    while ( s[i] != '\0')
        i = i+1;
    return i;
}
```

Note :
"a" → return as a string
'a' → return its ascii value.
"a" +10 → false (error)
'a' + 10 → Return ascii value

Assignment operators (=)

$i += 2 \longrightarrow i = i + 2$

Here value i+2 is assign to variable i.
There will be always a variable on the LHS.

Bit Operators

Every operater operates bitwise.

Ex A(60) = 0011 1100
B(13) = 0000 1101

A & B = 0000 1100 [Bitwise AND]
A | B = 0011 1101 [Bitwise OR]
A ^ B = 0011 0001 [Bitwise EX-OR]
~A = 1100 0011 [Bitwise NOT]

A<<2 = 11110000 (240) [Left shift by 2 bits]
A>>2 = 00001111 (15) [Right shift by 2 bits]

On Left shifting of n-bits number will change in $A \times 2^n$

In this case A=60 n=2
$A \times 2^n = 240$ [11110000]

In case of Right shift $\frac{A}{2^n} = \frac{60}{4} = 15$ [00001111]

Points to learn

Left Shift $A \times 2^n$

Right Shift $\frac{A}{2^n}$

Date :
Page No.

Ternary Operator

Ex① `(num%2 == 0) ? Printf("even") : printf("odd")`

- `(num%2 == 0)` → test condn
- `Printf("even")` → If condn is true
- `printf("odd")` → If condn is false

② `(a>b) ? max=a : max=b;`

This can also be written using if-else as

```
if(a>b)
  max=a;
else
  max=b;
```

So its better to use ternary operators.

Increment and Decrement Operators

`++n` → Increment before used (Pre-Increment)
`n++` → Increment after its value has been used (Post-Increment)

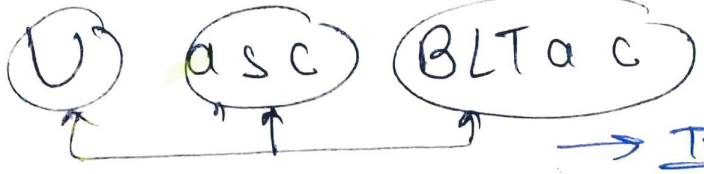
Precedence and order of evaluation of operators

Evaluate the operators according to precedence but if operators have same precedence then, evaluate it according to associativity.

Operators		Associativity
(), [] → .	Pointer content	L → R
!, ~, ++, --, +, -, *, &, sizeof, (type)	Unary operators	R → L
*, /, %	Arithmatic	L → R
+, -	Arithmatic	"
<<, >>	Shift operator	"
<, <=, >, >=	Comparison operator	"

Logical NOT

~ → 1's complement



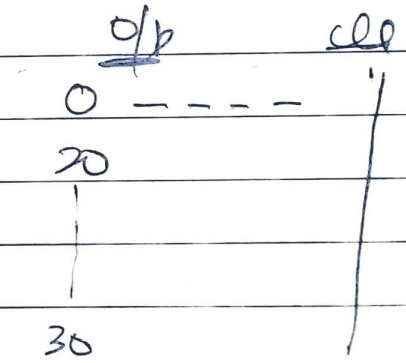
→ Trick for Table

Date :
Page No.

==, != } Comparison	L - R
& } Bitwise	"
^	"
|	"
&& } Logical	"
||	"
?: } ternary	"
=, +=, -=, /=, %=, *=, } Assignment/ Shorthand	R → L
^=, |=, <<=, >>,	R - L
, } Comma	L → R

Write a program to print fahrenheit - Celsius Table

Comment //* ... *//




```
#include <stdio.h>
main() {
int far, cel, low, upper, step;
    lower=0 ;
    upper=300;
    step= 20;
  far = lower;
while (far <= upper){
   cel= 5 * (far -32)/9;
printf ("%d\t %d\n", far, cel);
   far = far+step;
}
```

tab → ____ (4 spaces)

Additional Information regarding (Module 1)

If number > 0 positive
number = 0 not positive nor negative
number < 0 negative.

Short circuit Property
if (n==1 || j==1)
↓ T then dont need to check ↑

Right Associativity
(Right to left)

Left Associativity
Left to right

fmod(x,y) → returns the modulus after division and also works on floating point number.

→ Post increment/decrement operator have higher precedence than pre increment/decrement

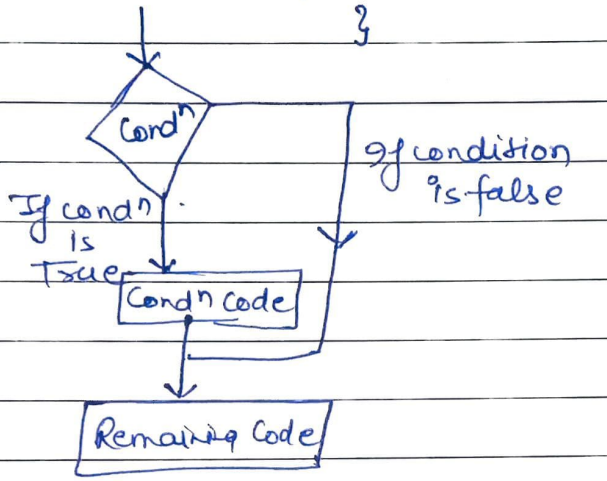
Module (2)

Date :
Page No.

Flow Controls

(1) If statement

```
if (boolean expression) {
/* statement will execute if boolean expression is true */
}
```



(2) if-else statement

```
if (BE1) {
/* statement will execute if BE1 is true */
}
else if (BE2) {
/* BE2 true but BE1 is false */
}
else {
/* Both BE2 and BE1 are false */
}
```

Note → else without if is an error but not vice-versa

(3) Write a program to check a whether a given number is even or odd.

Contn...

```
# include <stdio.h>
int main() {
    int num;
printf ("Enter a number");
 Scanf ("%d", &num);
  if (num % 2 == 0)
 printf ("num is even");
   else
    printf ("num is odd "); return 0;
}
```

Switch statement

Two cases doesn't have the same number.

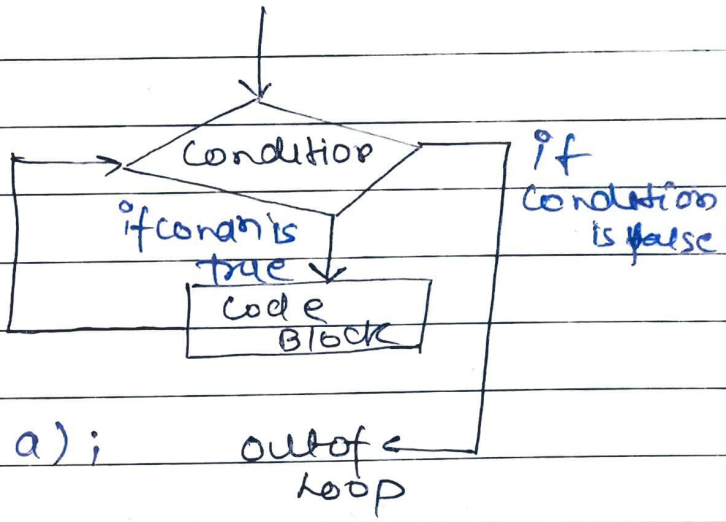
```
# include <stdio.h>
 void main() {
  int weekday;
 printf ("enter weekday");
  Scanf ("%d", &weekday);
Switch (weekday) {
 Case 0 : Printf ("Monday"); break;
 Case 6 : printf ("Sunday"); break;
 default : printf ("enter valid input");
}
```

Switch (input) → always an integer.

Break statement stop the execution of other preceeded statement otherwise all will be execute.

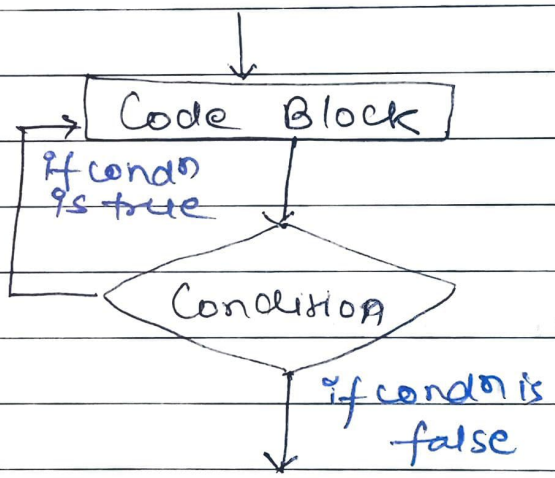
while Loop :-

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



Do-while Loop :-

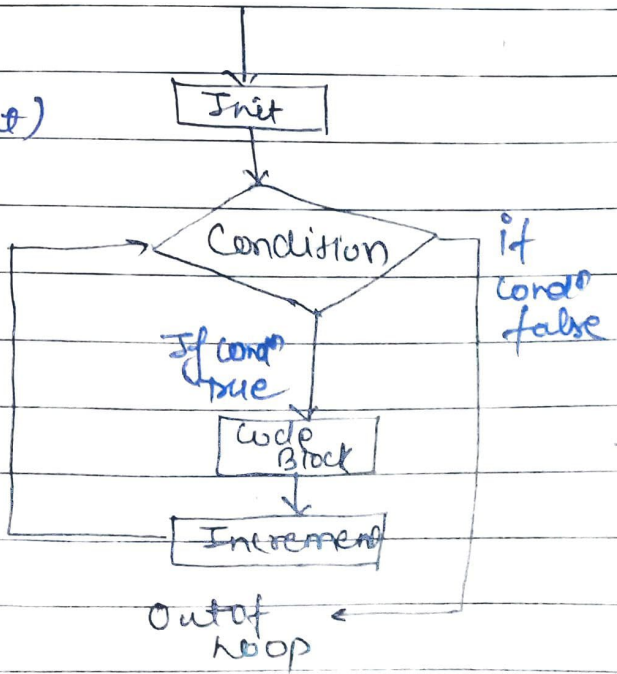
```
int main() {
    int a=20;
  do {
  printf ("a value : %d", a);
    a++;
  } while (a<20);
  }
```



for loop :-

```
for (init ; condn ; increment/decrement)
  {
    conditional code;
  }
```

check
move here
if true
move here



Write a program to read input until user enter a positive integer.

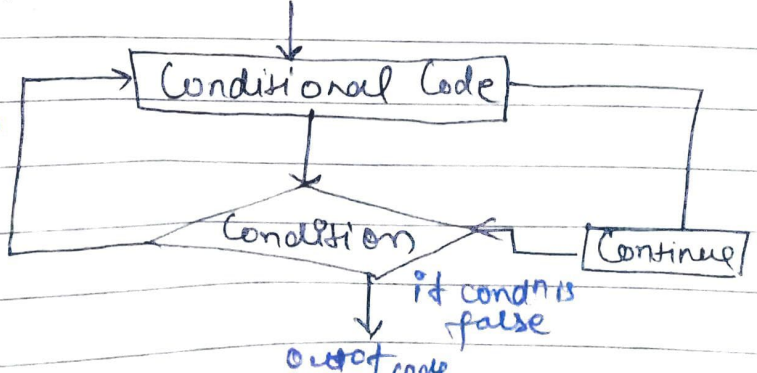
```
#include <stdio.h>
int main(){
    int n;
    do{
        printf("Enter a value");
        Scanf("%d", &n);
    } while (n<=0);
printf("n value : %d", n);
    return 0;
}
```

WAP to print the sum of only positive integers:-

Continue statement

```
void main(){
  int n, i, 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 integer %d", sum);
  }
```

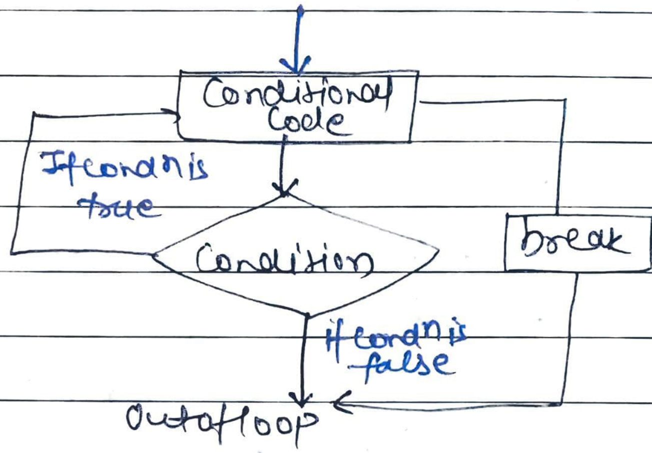
Control transfer to Increment → But in while loop transfer to condition.



Break Statement

WAP to read integers until user enters a negative integer or number of integer 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;   —— // stop the execution of loop and
                 transfer the control out of loop //
} }
}
```



WAP to check whether a given number is prime or not.

Prime no के लिए 1 और n को अलावा किसी से divisible नहीं पर हमने 1 लिया नहीं और Loop √n तक जाएगा नहीं तो अगर एक बार भी devide हुआ मतलब Prime नहीं है

```
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", n);
}
```

normally we use i<n but here we use $i <= \frac{n}{2}$ bcoz,

Ex if n=24

1, 2 ——— 23, 24

2×12
3×8
4×6

normally all the factors are lies before $\sqrt{n}$ = 4.something

So if no factor obtain before $\sqrt{n}$ no-one will found after $\sqrt{n}$ and $\sqrt{n} <= n/2$ always so we run the loop till n/2.

since $n/2 < n$ reduce space complexity.

WAP to find factorial of a given number.

```
void main (){
    int n, i;
unsigned long long fact = 1;
    printf ("Enter an integer");
    Scanf ("%d", &n);
    if (n<0)
    printf ("factorial doesn't exist");
    else {
     for(i=2;i<=n;i++)
       fact *= i;
     printf ("factorial of %d = %llu", n, fact); }
   }
}
```

%llu → unsigned long long

WAP to print half pyramid using *.

```
void main(){
    int i, j, n=4;
for (i=0; i<n; i++) {
  { for(j=0; j<i; j++){
      printf("*"); }
  }
     printf ("\n"); }
}
```

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

WAP to count no. of digits in an integer.

```
Void main() {
  int n, count =0;
printf ("Enter an integer");
  Scanf ("%d", &n);
  while (n)=0)
     n=n/10;
     ++count; }
printf ("no. of digits : %d",
        count); }
```

Date :
Page No.

WAP to check whether the given number is armstrong or not.

```
#include <math.h>
void main(){
int num, originalnum, remainder, result=0, n=0;
printf("Enter an integer");
Scanf("%d", &number);
original number = number;
while(originalnum != 0){
    originalnum /=10;
        ++n;
}
Original num = number;
while (originalnum !=0) {
    remainder = originalnum % 10;
    result += pow(remainder, n);
    originalnum /=10; }
(result == number)? Printf("Armstrong") : Printf("Not Armstrong");
}
```

Ex → $371 = 3^3 + 7^3 + 1^3$.
(371 is an armstrong number)

$12 = 1^2 + 2^2$
(12 is not an armstrong number)

WAP to print a star-pattern

```
(4-1)   _ _ _ *
(4-2)   _ _ * * *
(4-3)   _ * * * * *
(4-4) * * * * * * *
```

no. of spaces = total rows - row no.

no. of stars = (2 * row no.) - 1

```
void main() {
  int i, j, k, n = 4
  for (i=1; i<=4; i++) { # four 4 rows
    for (j=i; j<n; j++)
      printf (" ");
  }                          # To print spaces
  for (k=1; k<(i*2); k++) { }  # To print star
      printf ("*");
  }
  printf ("\n");
  }
}
```

WAP whether given number is palindrome or not.

Ex- 121, 323, 292

```
Void main(){
    int n, remainder, original, reversedNumber = 0;
  printf ("Enter a number:");
   Scanf("%d", &n);
    original = n;
   while (original != 0){
    remainder = n%10;
   reversedNumber = reversed number * 10 + remainder;
       n/=10; }
(original == reversedNumber)? printf("Palindrome"); printf("Not a
                                                    palindrome");
  }
```

WAP to generate a fiboracci series?

Given: first and second number of sequence.

```
void main(){
int first, second, sum, num, count=0;
 printf("Enter the no. of terms");
  Scanf("%d", &num);
 printf("enter first number");
  Scanf("%d", &first);
 print("enter second number");
  Scanf("%d", &second);
print("In fibonacci:%d%d", first, second);
  while(count<num){
        Sum = first + second;
        printf("%d", sum);
        first = second; second = sum;
          count++;
     }
```

Some other loop programs

→ Conversion of number. (Decimal to Binary type)

If Continue comes in a loop then

① while() → In while if continue comes returns to ( ) condition

② for() → In for loop returns to increment.

③ do{ while() → Returns to condition().