0111 1000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000

⑥ Sign bit (0) ⇒ No. is positive
Now the standard formed is.

0100 0000 0011 0111 1000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000

Session - 17:-

Operators :-
Function Data passing
Inc and Dec.

✗ - All predefined function declerations are present in header file.
✗ - All predefined function definition present in libraries.

Functions (2 types)
predefined
user defined

→ Ex:- main ()
{
int i, j;
printf("i = %d j = %d\n", i, j);
}

// Here main() calls printf, by passing 3 arguments.

1st argument
2nd 3rd

Note:- Functions data passing is done from right to left.

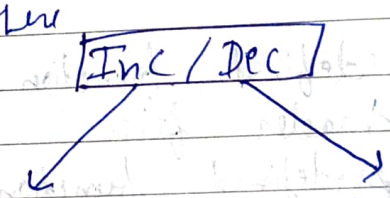
Ex:- void main() // ① 35 > 40 = 0
② 50 == 35 ③ k = 50
{
int k = 35
printf("%d %d %d \n", k == 35, k = 50, k > 40);

(data passing right to left but while printing it prints left to right.)

o/p:- 0 50 0

⇒ Increment and Decrement operators (++, --)

→ Unary operators.
→ These operators modify the operands, so we can supply only variables.
→ Two types are there



Pre Inc/Dec → operator 1st then operand 2nd
eg:- ++ □ □ ++i
-- □ □ --i

post Inc/Dec :- operand 1st the operator 2nd
□ □ ++ i++
□ □ -- i--

++ 10
-- 10
10 ++
] there compile time translation errors

Here, i is increment by one value

① int i = 10; i = i + 1; ⇒ i = 11

② int i = 10; ++i; ⇒ i = 11

③ int i = 10; i++; ⇒ i = 11.

① Here, 2 operations are to be performed ADD and MOV instructions are to be done. Here, the compiler may generate more instruction which generate more opcodes which slows down the execution as more opcode involves more machine cycle.

⇒ Here in this scenario '++i' is preferable and is faster.

→ Now for incrementing more than one value ADD and MOV instruction is preferable & is faster -

eg:- int i=10
i = i+1;
①

int i=10
++i;
++i;
++i;
++i;
++i;
②

Here ① is convenient.

→ Ex:- // Pre Inc operators

```
void main()
{
  int i=10;
printf("i=%d\n",i); // i=10
  ++i; // i becomes 11 here
  printf("i=%d\n",i);
}
```

o/p

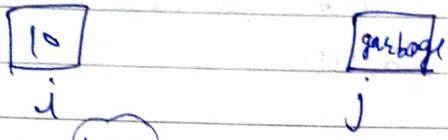
i=10
i=11

→ // Post Inc operator

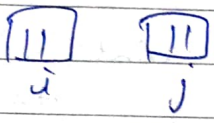
```
void main()
{
printf("i=%d",i);
  int i=10,j;
printf("i=%d\n",i);
  i++;
printf("i=%d\n",i);
}
```

- Rule of pre-Inc is first Int it and then assign it.
- Rule of post-Inc is first assign it and then Inc it

→ Initially,



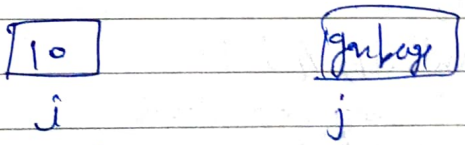
Now `j = ++i;` // This is incremented by 1 and this i value is stored in j then



o/p:- i=11, j=11

→ Consider,

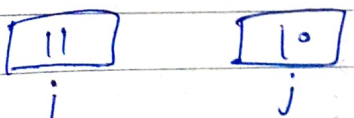
```
int i=10;
printf("%d\n", i);
j = i++;
printf(" i=%d  j=%d\n", i, j);
```



`j = i++;` ← As per rule of post Inc, 1st assign

initially i value is 10 → this 10 assigned to j.

After assigning j's value, i is incremented



o/p:- i=10
i=11 j=10

→ Behaviour of Inc and Dec operator in printf()

Ex:- void main()
{
int i=10;
printf("%d %d %d\n", i++, i++, i++);
function data passing

(i)++, (i)++, (i)++
12 11 10
then Inc then Inc then Inc

o/p:- 12, 11, 10

- Now consider.
printf("%d %d %d\n", ++i, ++i, ++i);
i=10
++(i), ++(i) ++(i)
13 12 11

this is the expected o/p 13 12 11

Note:- For Pre-Inc the compiler assigns the latest value.
So the o/p is
o/p:- 13, 13 13

Note:- For pre-Inc & no-inc, the latest value is going to be updated.

- Now consider:-
printf("%d %d %d %d %d\n", i++, ++i, i, i++, ++i);
i++, ++i, i, i++, ++i
13 13 12 11 11

13 13 12 11 11 expected o/p

But the o/p is,

o/p:- 13 14 14 11 14

This is because, the updated values are assigned to pre Inc & no Inc.

Session 18

→ Control statements
  → Comma operator
  → if
  → else if

- printf() returns the no. of printable characters
  eg :- x = printf("set\n");
  printf("%d\n", x);
  Here the output is 4
  Because inside the printf("set\n"); there are 4 characters
  ↑↑↑ ↑

Ex:-
```
void main()
{
int i=123, j;
j=printf("i=%d\n", i);
printf("j=%d\n", j);
```

o/p:- i=123
j=7
j is calculated as
i=123 \n
↑↑ ↑↑↑↑ ↑ → 7

→ Comma (,) operator:-

→ Comma acts as an operator and also as a separator.
→ In variable declaration and in a function call it acts as separator.

eg:- int i=10, j=20, k;
printf("i=%d j=%d k=%d\n", i, j, k);

→ In expression it will acts as an operator.
→ Comma is a binary operator.
(e1), (e2)
→ Result of comma (,) operator is 2nd operand
The result will be the right most one.

Ex:- Comma operator usage:-

```
void main()
{
int i;
printf("i=%d\n", i); //30
i = 10, 20; // Here comma acts as an operator.
printf("i=%d\n", i); //10
i = (10, 20)
printf("i=%d\n", i); //20
}
```

— Assignment operator has the higher priority than the comma operator.
⇒ i = 10, 20
→ So the value of i=10 is copied in i & there is no use of comma
→ Now i = (10, 20)
Here we have grouping have higher priority

i = (10, 20)
— for comma operator result is the 2nd operand.
— If i = (10, 20, 30) so, L to R
① 10, 20 & 20, 30 i = 30
20 3

Ex:- #include<stdio.h>
void main()
{
int i=10, j=20, k;
k=i=100, j=200; //comma acts as a operator
printf("i=%d j=%d k=%d\n", i, j, k);

k=i=100, j=200

100 is assign to i, the value of i is assigned to k.

o/p:-
i=100 j=200 k=100

Now consider
int i=10; j=200, k ~~=100~~;

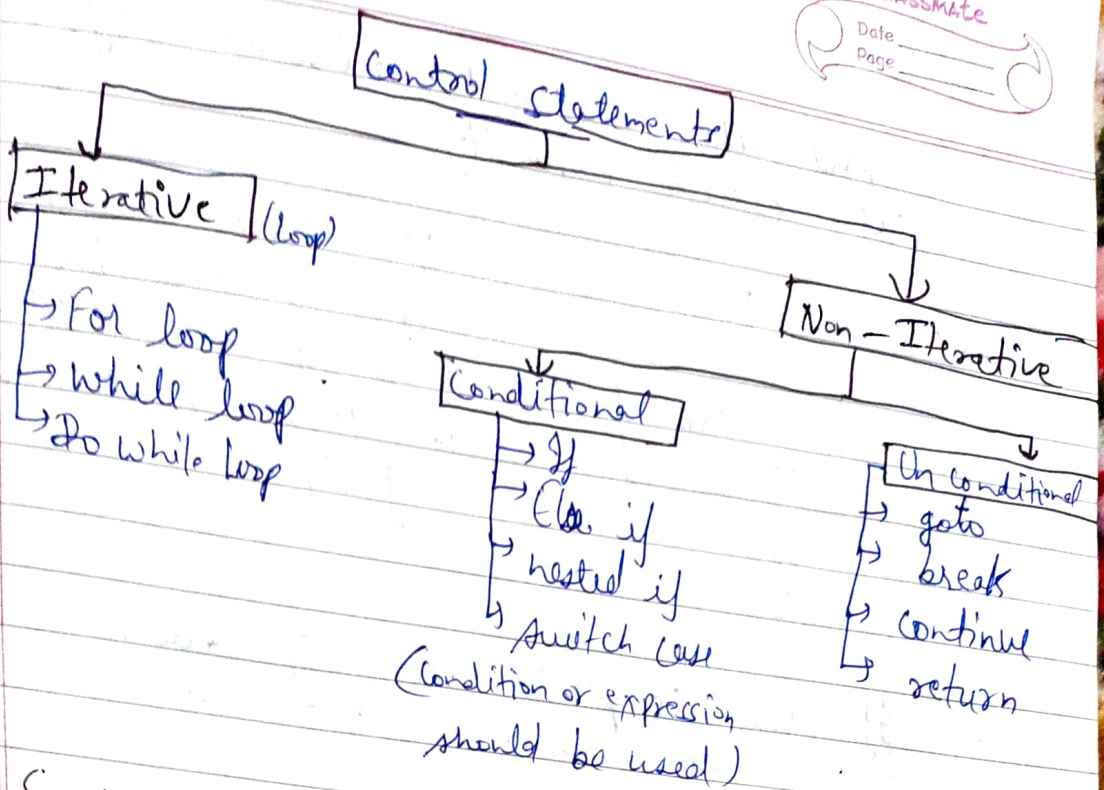
k=(i=100, j=200);
Two grouping should be solved

⇒ (i=100, j=200)
This is the result

k=200
printf("i=%d j=%d k=%d\n", i, j, k);
// 100 200 200
o/p:- i=100. j=200 k=200

→ Control statements :-

→ By default the statements are executed in sequence, one after another.
But this can be modify by using control flow structure.



Session 19:-

Control statements:-(if)

→ if

Non-Iterative conditional control statements:-

if:-

Syntax:

```
if (expression)
{
    s1
    s2      } Statements
    s3
}
```

Body of if

- Statements above the if executes normally.
- Now inside the 'if', if the result of the expression is non-zero (TRUE), the statements execute normally.
- When the 'if' condition is false the statements inside the 'if' won't execute.

And the statements below the 'if' block will execute normally.

→ Ex:-

```
S
S1
S2
if(exp)
S4
S5
S6
```

If there are no open & close braces, the next statement will be considered as the part of 'if'.

- If the (exp) is zero, S_4 does not execute
- If (exp) is non-zero S_4 executes.

Now:

```
Note:  S1
part of 'if'  S2
              S3
              if(exp);   → This is treated as dummy 'if'
              S4
              S5   ] Not a part of 'if'
              S6
```

This will be treated as

```
S1
S2
S3
if (exp)
{ no statements }
S4
S5
S6
```

∴ Even though a dummy 'if' is given, the expression inside the 'if' will be executed.

- Ex:-

```
S1
S2
S3
if(exp1, exp2, exp3)
S4
S5
S6
```

Here the execution of S_4 depends of the result of exp3 [Comma operator priority]

```
S1
S2
S3
if (exp);  → Treated as
{              dummy 'if'
S4 }  Not belongs to
S5            if
S6 }
S7
S8
```

as like

```
S1
S2
S3
if(exp)
{ No statements
}
S7
S8
```

Ex:- #include <stdio.h>
void main()
{
int i=10, j=20;
printf("Hello...\n");
if (i<j) // True so Hai is printed.
printf("Hai...\n");
printf("Bye\n");

o/p:- Hello...
Hai...
Bye

Now:-
if (i>j) // False Hai... is not printed

o/p:- Hello...
Bye.

Now:-
if (i>j); // dummy if

o/p:- Hellow...
Hai...
Bye

Consider :- // expression is going to be solved every time whether it is a dummy or not.

```
{
int i=10, j=20
if (i=j);
pf("Hai-\n");
pf("Hello\n");
pf("%d %d\n",i,j);
}
```

o/p :- Hai
Hello
20 20

→ 'if' is neither a function nor an operator
→ 'if' is a control statement
→ if does not return anything.

→ write a program to scan a character from the user display that character and its ASCII values, only if it is small letter / alphabet.

```
→ #include<stdio.h>
  void main()
  {
    char ch;
  printf("Enter the char\n");
  scanf("%d",&ch);
  if(ch>=97 && ch<=122) // if(ch>='a' && ch<='z')
  printf("ch=%c and its ASCII=%d\n",ch,ch);
  printf("Thankyou");} // if the 'if' is true then execute
```

o/p :- Enter the char..
c
char = c and its ASCII = 99
Thank you

o/p :- Enter A
Thank you

→ write a program to scan a character from the user display that character and its ASCII value, only if it is ~~small~~ letter upper case letter.

```
{
  scanf("%d", ch);
  if(ch>=97 && ch<=122)
    ch = ch - 32;
  printf("ch = %c and its ASCII = %d \n", ch, ch);
  printf("Thankyou\n");
    }
```

Here	128	64	32	16	8	4	2	1	→ Here the only difference is 5th bit
	7	6	5	4	3	2	1	0	
a=97	0	1	1	0	0	0	0	1	→ If 5th bit set then it is small
A=65	0	1	0	0	0	0	0	1	→ If 5th bit clear then it is clear capital

Small to capital → clear 5th bit Capital to small → set 5th bit

Small to capital

```
ch = ch & ~(1<<5);
      or
ch = ch & ~32
```

Capital to small

```
ch = ch | 32;
      or
ch = ch | 1<<5;
```

— Convert capital to small & vice-versa (Toggle)

```
ch = ch ^ 32
```
(1<<5) result is 32

```
ch = ch & ~(1<<5)
ch = ch & ~32
```

o/p:- Enter the char
a
Ch = A and its ASCII = 65
Thankyou.

Session → 20

→ Control statements
→ else if
→ else if ladder
→ nested if

else if :-

```
if (exp)
statement;
else
statement
```

```
if (exp)
{ s1
  s2
  s3 }
else
{ s1
  s2 }
```

when True ①

When False ②

Ex:-
```
#include<stdio.h>
void main()
{
int i=10, j=20
printf("Hi...\n");
if (i>j)                    // if (i<j)
printf("i is greater\n");   //o/p i is greater.
else
printf("j is greater\n");

printf("Thank you");
}
```

o/p:- Hi...
j is greater
Thankyou

- Between if else, you should not write any statement, which creates error
This show else without if error.

```
if(i>j)
printf("Hi\n");
printf("Hello\n"); // This creates error
else                    // Syntax (Translator error)
printf("Thankyou");
```

→ we need to display the character in upper case or lower case:-

ASCII
65 - 90 - capital
97 - 122 - lower
48 - 9 → 57

```
#include<stdio.h>
void main()
{
char ch;
printf("Enter the character.\n");
scanf("%c",&ch);
if(ch>='a' && ch<='z')
printf("small letter\n");

else if(ch>='A' && ch<='Z')
printf("Upper letter\n");

else if(ch>='0' && ch<='9')
printf("digit\n");

else
printf("special character");
}
```

Syntax of else if

```
else if

if (exp)          //when if is True not
 {                  going to checked else if
 }
else if
 {
 }
else if
 {
 }
else //optional
```

→ #Student data base program calculating the marks

① (0 – 39) Fail ② (40–59) C grade
③ (60 – 74) B grade ④ (75 to 100) A grade.

```
#include<stdio.h>
void main()
{
  int m
if (m < 40)
printf("FAIL\n");

else if(m < 60)
printf("C grade\n");

else if (m < 75)
printf("B grade\n");
```

else
printf("A grade");
}

Nested if:-

```
if (exp)
  {if( )
    { }}
   else
     { }
else {
     }
```

```
if ( )
{
  if ( )
  { if ( )
    {
    }
  }
```

→ Acc. to our need we can use any number of if statements.

Session 21

→ More about else if ladder
→ goto statement

→ As per C89 standard, 15 levels of 'if else' are supported.
→ As per C99 standard, 127 levels of 'if else' nesting can be used.

Whenever a program loaded into RAM for execution, there are three files which are opened.

① Stdin
② Stdout
③ Stderror

[There are also buffer.

→ scanf() will scan the data from stdin buffer.
→ printf will print the data in stdout buffer

When the user type the data, the data goes into the stdin buffer.

eg:-
```
char i, j;
pf("Enter i\n");
sf("%c", &i); // This is scanned by user buffer is empty
pf("Enter j\n"); //here
sf("%c", &j); // This scanf is not wait for the
pf("i=%d j=%d\n", i, j); // another value, it
                         consider \n as the other
                         value.
```

o/p:-
Enter i : //ASCII value of a
Enter j : j=10 → \n = 10
i=97

Note:- After scanning any datatype, if any character is scanned next, provide space " %c".

This bug occure only when scanning a character

```
scanf(" %c\n");
      ↑
    here space
```

→ Write a program to display the given no. is power of 2 or not.

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

```
printf("Enter the num \n");
scanf("%d", &num);
```

// when, num & num-1, the result of exp is '0' so it is power of '2'

```
if (num % 2 == 0)
{
    if ((num & (num-1)) == 0)
    printf("Power of 2 \n");
    else
        printf("Even \n");
    }
else
printf("odd \n");
```

Q → Write a program to scan a no. and bit position from the user to display in the no. the bit is set or clear.

```
{
void main()
int num, pos;
printf("Enter the number \n");
scanf("%d", &num);
printf("Enter the position \n");
scanf("%d", &pos);

if (pos >= 0 && pos <= 31) // int range (0-31) bit (1 byte)
  {
    if (num & 1<<pos)
        printf("set \n");
    else
        printf("clear \n");
  }
else
  printf("wrong bit position \n");
}
```

→ When the user provide wrong bit pos, the program is terminated.
So, we are using 'goto' statement, to go to the bit pos to enter the right value.

goto :-

→
```
{ int num,pos;
pf("Enter number\n");
sf("%d",&num);

L1:

pf("Enter pos (0-31)\n");
sf("%d",&pos);

if (pos>=0 && pos<=31)
  {
   if (num & 1<<pos)
    pf("set\n");
   else
    pf("clear\n");
  }
else
  {
   pf("Wrong bit pos\n");

   goto L1;
  }
}
```

// To restrict the no. of wrong attempt, set up a counter.

considering the same program

```
{
 int num, pos, c=0;        // C=0 for counter
 pf("Enter no \n");
sf("%d",&num);
 L1:    // Setting a label Here
pf("Enter the bit pos (0-31) \n");
sf("%d",&pos);

if(pos>=0 && pos<=31)
 {
   if(num & 1<<pos)
pf("set \n");
  else
  pf("reset \n");
   }
else
 {
   c++;// Counter value increased here
  if(c<=2)
 {{
  printf
   if(c==2)
   printf("Last chance: Warning \n");
   else
    pf("Invalid position \n");
     goto L1;
 }
    }
```

goto statement :-

→ goto is unconditional non-iterative control statement.

→ goto can be used as forward jump or backward jump.

→ jumping can be possible within a function (local jump)

eg:-

```
main
{
    ≡
    goto L1;
    ≡
    L1;
```

Forward jump

```
main
{
    ≡
    L1:
    ≡
    goto L1;
```

Backward jump

Syntax:-

1. Set the label
2. call the label

① To set Label

Label name:

② To call the label

goto labelname;

- Variable name and label name can be same id.

eg:-

```
int L1;
pf("Hello\n");
L1:
sleep(1); <- //A delay of 1 sec
goto L1; pf("Hai\n");
goto L1;
pf("bye\n");
}
```

o/p:-

```
Hello
Hai
Hai
Hai
:
:
```

Now consider

```
Void main()
{
  Pf("Hello\n");
  goto L1;

  sleep(1);
  Pf("Hai\n");

L1:
}
Pf("bye\n");
```

o/p:-
Hello
bye ...

Session 23

→ Control statement
  → goto
  → for loop

→ Below the label, a minimum of one statement should be there, when we use are setting the label at the end of the function.
This rule is for the forward jump

```
Void main()
{
  Pf("Hai\n");
  Pf("Hello\n");
  goto L1;
  Pf("bye\n");
L1:
}  // This thrown error (no statement)
```

write:

```
goto L1;
pf("Bye");
L1:
; //using this we can avoid the previous
}          error.
```

Write a program to display multiplication table using goto.

```
#include<stdio.h>
void main()
{
int num, i=1;
pf("Enter the number\n");
scanf("%d", &num);
L1:
pf("%d * %d = %d\n", num, i, num*i);
i++;    // ++i can be used
if (i<=10)
goto L1;
}
```

o/p

```
Enter the number
5
5* 1 = 5
5* 2 = 10
:
5* 10 = 50
```

Write a program to print the binary format of a given number using goto.

Generally a given no. is stored in its binary format

eg: int i=10 i=10 is stored as

00000000 00000000 00000000 00000000

printing the number into binary and not converting it to binary

Here we need to check, whether each bit is set or clear from 31st to 0th bit systematically

```
#include<stdio.h>
void main()
{
int num, pos=31; // if '0' is initialise, the bits will be printed from reverse order
pf("Enter the number\n");
sf("%d", &num);
L1:
pf("%d", num>>pos&1) // has the o/p possibility of '0' and '1'
pos--; // pre-dec works
if(pos>=0)
goto L1;
pf("\nThanks\n");
}
```

o/p Enter the num
10
00000000 00000000 00000000 00001010
Thanks..

To make code more generic

```
void main()
{ int num, pos;
pf("Enter the number\n");
sf("%d", &num);
pos = sizeof(num)*8-1;
L1:
pf("%d", num>>pos&1);
pos--;
if(pos>=0)
goto L1;
pf("Thanks\n");
}
```

o/p:- 00000000 00000000 00000000 00001010
Thanks

To print the space after 1 byte (8bit)

```
if(pos%8==0)
pf(" ");
pos--;
```

o/p:- 00000000 00000000 00000000 00001010
Thanks