Flow Control

Date

Selection Statement

1. if-else
2. Switch()

Iterative Statement

1. while()
2. do-while()
3. for()
4. for-each loop

1.5 V

Transfer Statement

1. Break
2. Continue
3. return
4. try-catch-finally
5. assert

1.4 V

if-else

Syntax :

any other type then C.E

should be boolean Type

```
if(b)
{
  Action if b is true
}
else
{
  Action if b is false
}
```

Date

```
int x=0;
if (x)
{
 Sopln ("Hello");
}
else
 { Sopln ("Hi");
 }
```

only boolean
C.E

```
int x=10;
if (x=20)
{
 Sopln ("Hello");
}
else
 {
  Sopln ("Hi");
 }
```

Comparison operator
missing
C.E

```
int x=10;
 if (x==20)
        comparison here
  {
   Sopln ("Hello");
  }
  else
   {
    Sopln ("Hi");
   }
```

Output Hi

```
boolean b= true;
if (b= false)
 {
  Sopln ("Hello");
 }
else
 {
  Sopln ("Hi");
 }
```

Even though it is assignment but in the end the type is boolean

Output Hi

```
boolean b = false;
if (b==false)
 {
  Sopln ("Hello");
 }
else
 {
  Sopln ("Hi");
 }
```

Date

```
if (True)
    Sopln("Hello");
```

Valid ✓

```
if (true);
```

Valid java statement
output: No output

```
if (true)
    int x=10;
```

invalid ✗

```
if (true)
{
    int x=10;
}
```

Valid ✓

→ Else part and { } curly braces are optional. without curly braces, only one statement is allowed under if which should not be declarative statement.

Semi colon is a valid java statement which is also known as Empty statement.

```
if(true)
    if (true)
        Sopln ("Hello");
    else
        Sopln("Hi");
```

else always related to nearest if

Switch

Date _____

```
switch (x)
{
  case 1 :
        Action-1 ;
        Break ;
  Case-2  Action 2 ;
        break ;
  :
  Casen  Actionn
        break
  default :  default Action
}
```

1.4V	1.5V	1.7
byte	Byte	
short	Shot	String
char	Character	
int	Integer	
	+ enum	

wrapper class

~~boolean~~
~~long~~
~~float~~
~~double~~

1. The allowed argument types for the switch statement are byte, short char int
   until 1.4V
   but 1.5V onwards corresponding wrapper classes and enum type is also allowed
   ~~from~~ from 1.7V onwards string type also allowed

2. {} Curly braces are mandatory except Switch every where curly braces are optional

3.
```
Switch (x)
{  case 1 : sopln(1);
           break;
   default : sopln("def");
}
```
optional

Date

3. Without Case and default Switch is valid ie: both Case and default are optional.

4. Independent statement are not allowed
C.E : Case, default or } Expected
error

```
int x = 10;
Switch (x)
{
    Sopln ("Hello");
}
```

5

```
int x=10;
int y=20;
Switch (x)
{
    Case 10:
        Sopln(10);
        break;
    Case y:
        Sopln (20);
        break;
}
```

→ Not Constant

C.E : Constant Expression required

Every Case level should be Constant

ib final y=20; then no C.E

Perfectly Valid

```
int x=10;
final int y=20;
Switch (x)
{
Case 10:
Case y:
}
```

Valid

Date

6.
```
int x=10
Switch (x+1)
{ Case 10:
    Sopln(10);
    break;
Case 10+20+30;
    Sopln(60);
```

Expression → (int + int)

Expression

Both Switch argument and Case level Can be Expressions but Case level should Constant Expression

7.
```
byte b=10;
Switch (b)
{
Case 10;
    Sopln(10);
    break;
Case 100;
    Sopln(100);
    break;
Case 1000:
    Sopln(1000);
}
```

-128 to 127

C.E: Possible loss of precision found: int required: byte

```
byte b=10;
Switch (b+1)
{
Case 10;
    Sopln(10);
    break;
Case 100:
    Sopln(100);
    break;
Case 1000:
    Sopln(1000);
}
```

(byte + int) → (int)

Perfectly Valid

Spiral

Date

```
int x=10;
Switch (x)
{
  Case 97:
    Sopln (97);
    break;
  Case 98:
    Sopln (98);
    break;
  Case 99:
    Sopln (99);
    break;
  Case 'a':
    Sopln ('a');
}
```

CE: Duplicate Case Level

Summary:

Case Label

1. it should be constant expression
2. The value should be in the range of Switch argument type
3. Duplicate Case label are not allowed

Fall through Inside Switch

```
Switch (x)
{
  Case 0:
    Sopln (0);
  Case 1:
    Sopln (1);
    break;
  Case 2:
    Sopln (2);
  default:
    Sopln ("def");
}
```

x=0	x=1
0	1
1	
x=2	**x=3**
2	def
def	

Spiral

Date

Ex:2

```
Switch(x)
{
Case 1:
Case 2:
Case 3:
    Sopln("Q-4");
    break;
Case 4:
Case 5:
```

Within the switch if any case is matched from that case onwards, all statements will be executed until break or end of the switch This is called fall through inside switch

Advantage: We can define common action for multiple cases (Code - reusability)

Conclusion:

1. Within the switch we can take default case at most once
2. Default case will be executed iff and only if there is no case will matched, within switch we can write default case any where but it is recommended to write as last case

Date

```
Switch(x)
{
default :  → wait for some time
        Sopln("def");
Case-0:
        Sopln(0);
        break;
Case-1:
        Sopln(1);
Case-2
        Sopln(2);
}
```

x=0 → 0
x=1 → 1, 2
x=2 → 2
x=3 → def, 0

Iterative Statements

① while( ); → result set (don't know the record are in result set)

```
while(rs.next())
{
  .
  .
  .
}
```

```
while(e.hasMoreElements())
{
  ≡
}
```

→ enumeration (don't know the elements) in collection and cursor concept

9873981660 | 66

Date

While (itr.hasNext())
{
    we don't know the number of iteration in advanced
}

Syntax:
While(b)
{
    Action
}

should be boolean type only
other type → C.E

# While (1)
{
    Sop ln ("Hello");
}
int type
C.E incompatible type
found : int
required: boolean

• while(true)
Sop ln ("Hello");

• while(true);

• While (true)
int x=10;

• While (true)
{
    int x=10;
}

curly braces are optional and without curly braces we can take only one statement under while which not be declarative statement

# While(true)
{
    Sop ln ("Hello");
}
Sop ln ("Hi");
keep on executing
No chance for Sop ln ("Hi");
C.E unreachable statement

While(false)
{
    Sop ln ("Hello");
}
Sop ln ("Hi");
C.E unreachable statement
as total loop body unreached

int a=10, b=20;
While (a<b)
{
    Sop ln ("Hello");
}
Sop ln ("Hi");
output: Hello Hello ... infinite times

```
int a=10,b=20;
while (a>b)
{
    Sopln("Hello");
}
Sopln("Hi");
```

output: Hi

```
final int a=10; b=20;
while (a<b)
{
    Sopln("Hello")
}
Sopln("Hi");
```

CE
unreachable statement:
Sopln("Hi");

```
final int a=10, b=20;
while (a>b)
{
    Sopln("Hello");
}
Sopln("Hi");
```

CE unreachable statement:
{

Note: Every final variable will be replaced by the value at compile time only

```
final int a=10;
int b=20;
Sopln(a);
Sopln(b);
```

After compilation

```
Sopln(10);
Sopln(b);
```

```
final int a=10,b=20;
int c=20;
Sopln(a+b);
Sopln(a+c);
Sopln(a<b);
Sopln(a<c);
```

After compilation

```
Sopln(30);
Sopln(10+c);
Sopln(true);
Sopln(10<c);
```

Do{this;} while (condition matched);

Date

do-while ( )

While (a<b)
{
≡ Zero times
}

Syntax:

```
do
{
    body
} while (b);
```

→ mandatory

↘ Should be boolean type

```
Public class Test {
  p s v m(String[] args)
  {
    int count=1
    do{
      sopln("count:"+count);
      count++
    }
    while (count<=5);
    sopln("loop finished");
  }
}
```

output:
count : 1
count : 2
count : 3
count : 4
count : 5
loop finished

if we want to execute loop body at least once then we should go for do-while

Curly braces are optional and without curly braces we can take only one statement b/w do & while which should not be declarative statement.

```
do
sopln("Hello");
while (true);
```
✓

```
do;
while (true);
```
✓

```
do
int x=10;
while (true);
```
✗ declarative statement without curly braces

```
do
{ int x=10;
} while (true);
```
✓

```
do
while (true);
```
✗

no body
(at least should be one

Spiral

Date

no statement. only one statement

```
do while (true)
    Sopln ("Hello");
    while (false);
```

→ not declarative statement

Perfectly Valid

```
do
    while (true)
        Sopln ("Hello");
while (false);
```

output
Hello
Hello
:
:

```
int a=10, b=20;
do
{
    Sopln ("Hello");
} while (true);
    Sopln ("Hi");
```

keep on repeating CE: Unreachable Statement

```
do
{
    Sopln ("Hello");
} while (false);
    Sopln ("Hi");
```

output: Hello Hi

```
do
{
    Sopln ("Hello");
} while (a<b);
    Sopln ("Hi");
```

output: Hello Hello

```
int a=10, b=20
do
{
    Sopln ("Hello");
} while (a>b); false
    Sopln ("Hi");
```

output: Hello Hi

```
final int a=10, b=20;
do
{
    Sopln ("Hello");
} while (a<b);
    Sopln ("Hi");
```

↳ Unreachable statement.

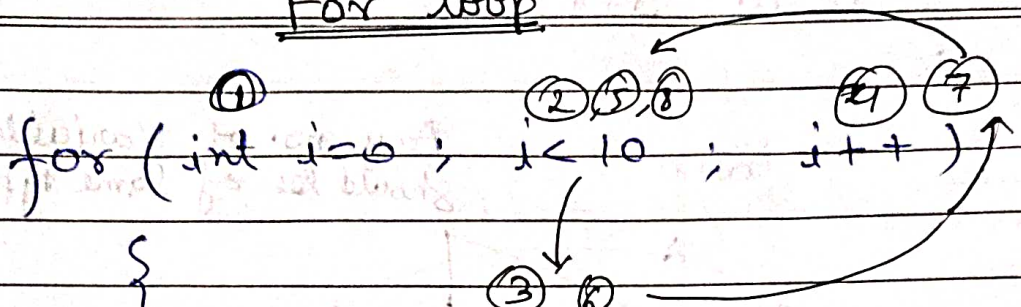
```
final int a=10, b=20;
do
{
    Sopln ("Hello");
} while (a>b);
    Sopln ("Hi");
```

op: Hello, Hi
Hi

Spiral

For loop

Date



SYNTAX:

```
for ( initialisation_Section; Conditional_check;
        increment_decrement_section )
{
    loop body
}
```

```
for (int i=0; true; i++)
    Sopln ("Hello");
```

Valid

```
for (int i=0; i<10; i++);
```

Valid

```
for (int i=0; i<10; i++)
    int x=10;
```

invalid

Curly braces are optional and without curly braces we can take only one statement under for loop, which should not be declarative statement.

Only one time, initialisation only local variable and declaration

Date

```
for (int i=0; i<10; i++)
{
}
```

also

Any no. of variable but should be of same type

✓ `int i=0, j=0;`

in

✗ `int i=0; String s="durga";` different data type variable get C.E

✗ `int i=0, int j=0;` Conditional check. then invalid

Complete:

```
int i=0;
for (sopln("Hello Boss U R Sleeping"); i<3; i++)
{
    sopln("No Boss U only sleeping");
}
```

Hello Boss UR sleeping
No Boss U only sleeping
No DOS U
No S

* In the initialisation section we can take any valid java statement including sopln.

: Conditional check :-

- `for (int i=0; i<j; i++)`

`(a<b && b>30 || x<l)`

any conditional statement but should be boolean Type

- If no condition is there by default True would be at conditional check

Date

```
int i=0;
for (System.Out.println("Hello you are Sleeping"); ; i++)
  {
    Sysopln ("No Boss U only Sleeping" +i);
  }
```

True by default

Output Hello you are Sleeping
No Boss U only Sleepin
No - - - - -
N - - - - - -

—: Increment & decrement section :—

```
int i=0;
for (Sopln ("Hello"); i < 3; Sopln ("Hi"))
  {
    i++;
  }
```

o/p Hello
Hi
Hi
Hi

```
for (int i=0; i<10 ; i++)
  {
  }
```

≡ All three parts of the loop are optional and independent

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

infinite times

```
for (;;);
```

Date

```
for (int i=0; true; i++)
{
  Sopln ("Hello");
}
Sopln Hi
```

C.E unreachable statement

```
for (int i=0; false; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

C.E unreachable statement

```
for (int i=0; true; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

C.E : unreachable statement

```
int a=10; b=20;
for (int i=0; a<b; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

o/p : Hello
Hello

```
int a=10; b=20;
for (int i=0; a>b; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

o/p : Hi

```
final int a=10, b=20;
for (int i=0; a<b; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

C.E unreachable statement

```
final int a=10, b=20;
for (int i=0; a>b; i++)
{
  Sopln ("Hello");
}
Sopln ("Hi");
```

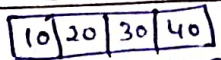
for-each loop (Enhanced for loop)

Date

1.5 Version

- It is specially designed loop to retrieve element of Array & Collections.

eg.1 To print elements of one-dimensional Array




```
int [ ] x = {10, 20, 30, 40};
```

Normal for loop

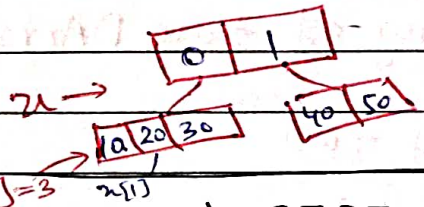
```
for(int i=0; i<x.length; i++)
{
    Sop ln x[i];
}
```

enhanced for loop

```
for (int x1 : x)
{
    Sop ln (x1);
}
```

OUTPUT: 10
20
30
40

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




```
int[][] x = {{10,20,30},{40,50}};
```

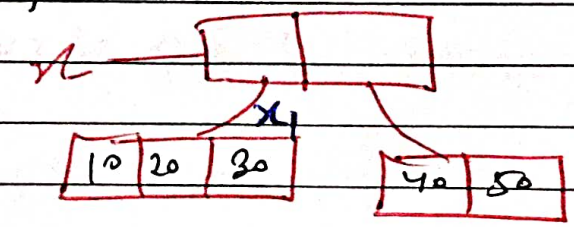
```
for(int i=0; i<x.length/2; i++)
{
    for(int j=0; j<x[i].length; j++)
    {
        Sop ln(x[i][j]);
    }
}
```

0<2

output

x[0][0] = 10
x[0][1] = 20
x[0][2] = 30
x[1][0] = 40
x[1][1] = 50

```
for(int[] x1 : x)
{
    for(int x2 : x1)
    {
        Sop ln(x2);
    }
}
```



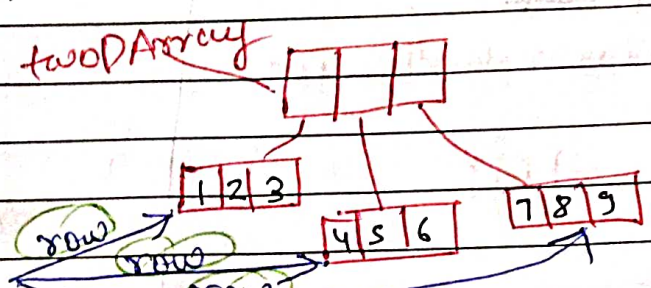
Spiral

#

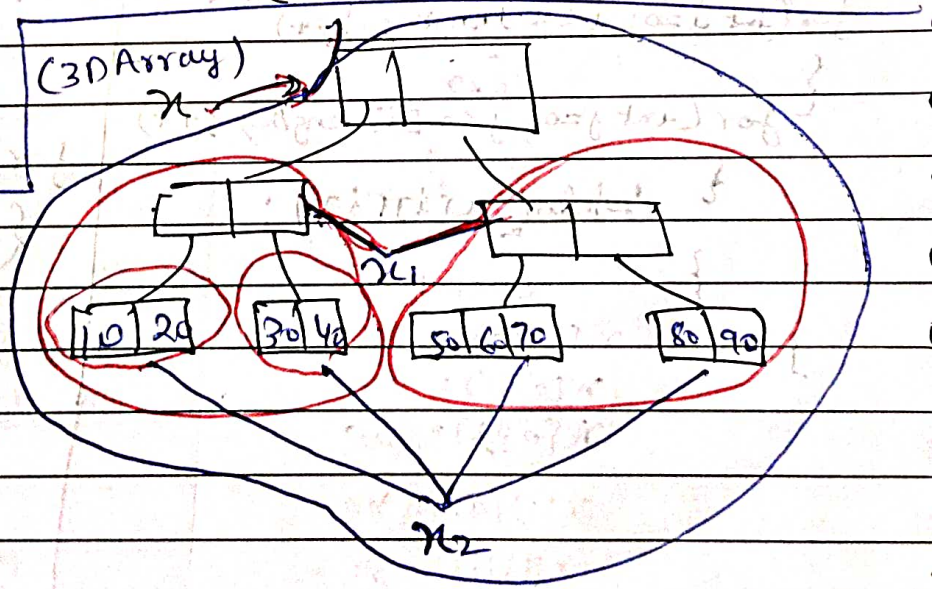
Date

```
public class TwoDimensionalArrayPrint {
    public static void main(String[] args) {
        int[][] twoDArray = {{1,2,3},{4,5,6},{7,8,9}};

        for (int[] row : twoDArray)
        {
            for (int element : row)
            {
                System.out.println(element);
            }
        }
    }
}
```




```
for (int[] row : twoDArray)
{   twoDArray ka Row
    For each one dimension array of twoDArray
    for (int element : row)
    {   For Each element of row
        System.out.print(element)
    }
}
```



#

for (int[][][] x.

Spiral

```
for (int[][] x1 : x)
{
    for (int[] x2 : x1)
    {
        for (int x3 : x2)
        {
            sysop (x3);
        }
    }
}
```

Date

```
for (int i=0; i<10; i++)
{
    sysop ("Hello");
}
```

Not using ~~Advan~~ for-each loop

We can't write an equivalent for-each loop directly.

for-each loop :- it is the best choice for retrieving Array & Collection.

Its limitations is applicable for only Arrays and Collections and it is not General purpose loop.

Loop Hole for Advanced for loop

```
int[] x = {10,20,30,40,50};
for (int i = x.length-1; i>=0; i--)
{
    Sopln (x[i]);
}
```

o/p: 50 40 30 20 10

reverse order

We can't write an Equivalent for-each loop directly

not possible in

- By using normal for loop we can either print Array Element in reverse order or original order
- By using for-each loop only we can print in original order

Spiral

Date

Iterable(I) Interface

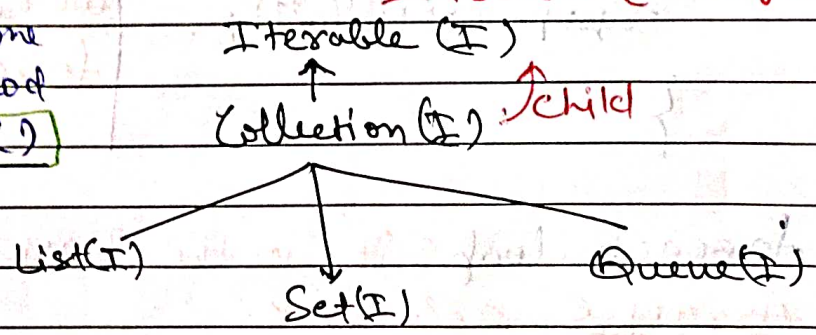
* target element can be Array / Collection

```
for (each item x : target)
```

↳ it should be iterable object
Said then

- java.lang.Iterable (Interface)
  - → 1.5 version
  - → only one method
    - iterator()

* Array & Collection are Interable (Interface)



Q. What is differences b/w Iterator and Iterable

Iterator(I)	Iterable(I)
1. it is related to Collections	• it is related to for-each loop
2. we can use to retreive the elements of Collection one by one	• the target element in for-each loop should be Iterable
3. java.util. package	• java.lang pkg
4. 3 methods hasNext() next() remove()	• 1 method iterator

TRANSFER STATEMENT

Date

* Break;

inside switch

```
int x=0;
Switch (x)
{
  Case 0:
      Sopln (0);
  Case 1:
      Sopln (1);
      break;
  Case 2:
      Sopln (2)
  default:
      Sopln ("def");
}
```

O/P: 0
1

Used to stop fall-through

absence of break
O/P: 0
1
2
def

inside loop

```
for (int i=0; i<10; i++)
{
  if (i==5)
    break;
  Sop(i);
}
```

O/P: 0
1
2
3
4

inside Labelled block

condition met break work

```
Class Test
{
  psv main (String [] args)
  {
    int x=10;
    l1:
    {
      Sopln ("begin");
      if(x==10)
        break;
      Sop ln ("end");
    } Sopln ("Hello");
  }
}
```

output: begin
Hello

Spiral

```
int x=10;
if(x==10)
    break;
Sopln("Hello")
```

C.E

Date

Note: if we using break; statement other than switch, loop or labelled block we will get C.E saying break outside switch or loop

break;

^

* Continue;

inside loop:

```
for(int i=0; i<10; i++)
{
    if (i%2==0)
        continue;
    Sopln (i);
}
```

Continue

* means within the loop skip current iteration and continue for the next iteration

skip

* If condition met ie; (i%2==0) 0==0 then continue work

output:

not skipped (1==0) print (i)

1
3
5
7
9..
:

skipped 0==0, 2%2==0, 4%2==0, 8%2==0

Condition met, skipped

inside switch Continue:

The 'continue' statement is used to skip the remaining code inside a loop iteration and move on to next iteration. However, within a 'switch' statement, there isn't a clear loop construct that the continue statement would refer to.

Labelled block:

Date _____

Same explanation as "inside switch".

```
class Test
{
  public static void main (String [] args)
  {  int x=10;
     if (x==10)
        Continue;
     System.out.println ("Hello");
  }
}
```

C.E : Continue outside of loop
Continue;

* Labelled Break & Continue;

Used when you want to break out of or continue a specific enclosing loop or block of code, especially when dealing with nested loops or complex control structure.

```
l1:
for (————————)
{
  :
  l2:
  for (————————)
  {
    :
    for (————————)
    {
      break l1; // block of code
      break l2;
      break;
    }
  }
}
```

≈ ≈ ≈ ≈

Continue l1;
Continue l2;
Continue;

Date

```
l1:
for (int i=0; i<3; i++)
{
    for (int j=0; j<3; j++)
    {
        if (i==j)
            break l1; // replace one by with the below
        Sopln (i+"..."+j);
    }
}
```

`break;`

1 .. 0
2 . 0
2 .. 1

`continue;`

0 .. 1

`break l1`

No output

`continue l1`

1. i=0 j=0 (0=0) break; (innerloop)
2. out for loop i++, i=1, 1<3 satisfied j=0
   (1==0) Sopln (o/p) 1 ... 0 (still inner loop)
3. j=1, 1<3 satisfied (1=1) break; innerloop
4. out for loop i=2 2<3 satisfied j=0
   (2=0) Sopln (o/p) 2 ... 0 (Still inner loop)
5. j=1 1<3 satisfied (1=2) ~~break; inner loop~~
   Sopln (o/p) 2 .. 1
6. ~~out for loop i=3 3<3 not satisfied~~

i = ~~0~~ + 2

j = ~~0~~ + ~~0~~ + 2

1 .. 0
2 . 0
2 . 1

for break 1:

Date

i=0

j=0

No output

for Continue;

~~i = 0 X 2~~

~~j = 0 X 2 3 0 X 0~~

i = 0 X 2 3

j = 0 X 2 3 0 X 2 3 0 X 2

output

0 ... 1 ~

0 ... 2 ~

1 ... 0 ~

1 ... 2 ~

2 - 0 ~

2 - 1 ~

for continue 1;

~~i = 0 X 2 3~~

~~j = 0 0 0~~

i = 0 X 2 3

j = 0 X 0 X 2

output

1 - 0 ~

2 - 0 ~

2 - 1 ~

* do-while() vs Continue (Dangerous combination)

```
int x=0;
do
{
    x++;
    Sopln(x);
if(x++<5)      if(++x<5)
                continue;
    x++;
    Sopln(x);
} while(++x<10);
```

x = 0 X 2 3

o/p
1
4
6
8
10

o/p = 1

In the expression if (x++ <5), the condition check will happen before the increment is applied to the variable 'x'

1. The current value of 'x' is used in the comparison ('x<5').
2. The comparison result (true or false) is determined based on the current value of 'x'.
3. After the comparison is done, the value of 'x' is incremented by 1 using the post increment operator ('x++').

Spiral