Chapter 5 :- Control statement

Control statement provides order in which various instructions in the program will be executed. Defines **how control is transferred between the program**.

5.1. Compound statement / Block

group of statements enclosed within the { }

```
{   statement 1
    -11-  2
    -11-  3
    -----
}
```

Block equivalent to single statement

5.2. If / Else

takes one of the two possible actions

```
if (expression)
        statement 1     action 1
else
        statement 2     action 2
```

expression = true
↳ statement 1 is executed
else
↳ statement 2 –

5.2.1 :- Nesting of If / Else

```
if (expression)
{   if (expression 2)
        statement 1;
    else
        statement 2;
}
else
        statement 3;
```

Note :- Notify nesting if-els clearly as it will create **Dangling Else Problem**

Program :- To find if year is leap year or not
A centinnal (÷ by 100) year is leap year if its divisible by 400
A non-centinnal is leap year if its divisible by 4

```
if (year %. 100 == 0)
{   if (year %. 400 == 0)        Centinnal
        leap year;
    else
        Not leap;
}else                            Not centinnal
{   if (year %. 4 == 0)
```

OR using single line

(Imp ***)

```
if (year %. 100 != 0
  ff year %. 4 == 20 ||
  year %. 400 == 0 )
```

leap year ;

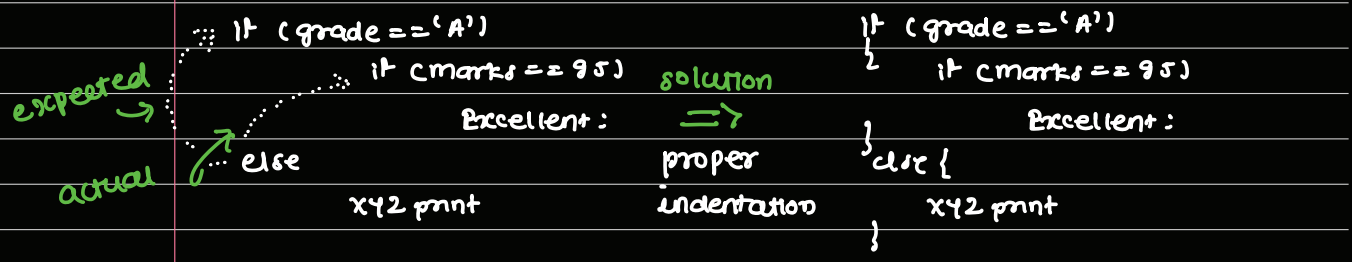
else

Not leap ;

}

5.4.2: Dangling Else Problem

- compiler always associates "else" part with closest associated "if" part for example



- got this problem coz of missing else in nested if statement
- use braces if -if-else creates confusion

5.4.3: if-Else Ladder

- if there is "If-else" pair in every else part except last

5.5. Loops

Loops → while loop, do while loop, for loop

5.5.1:- While Loop statements:

```
while (expression)
{
    statement 1
    statement n
}
```

Perform operation inside {} every time expression is true exit when false.

Program:- Program to print sum of digits of any number

```
while (number is not 0) {
    rem = number % 10 ;   taking last digit
    sum = sum + rem;
    number = number / 10 ;
}
```

Program:- Program to find factorial of number $n! = n \times n-1 \times n-2 \times \dots \times 2 \times 1$

```
while (n >= 1) {
    fact = fact * n      (fact=0 initially)
    n--;
}
```

Program Binary to decimal

```
while (binary_no > 0) {  rem = b_no % 10;
                         dec_no = decno + rem x 2^dtt
                         binary no = binary no / 10; }
```

5.5.2 :- **Do - while Loop :-**

```
do {
    statement 1
    :
    statement n
} while (condition);
```

Loop body executed first & then condition is evaluated. Guarteed to execute at least once apart from while loop.

5.5.3 :- **For Loop statements**

```
for ( expression 1, expression 2, expression 3 ) {
    statement :
    :
    statement : }
```

expression 1 → initialisation (executed only once)
expression 2 → test
expression 3 → update
(can be single expressions or multiple expressions)

equivalent while loop format :-

```
expression 1 :
while (expression 2) {
    statement 1
    :
    expression 3; }
```

Program :- multiply two +ve integers without * operator

```
for ( i=num1; i<num2 ; i++)
        result= result + num1 ;
```

Program :- sum of series up to N terms 1+2+3+------ +n

```
for (i= 1 ; i<n  ; i++) {
        sum= sum+ term ;
        term= term+ i ; }
```

(Imp +++)

Program :- febonacci series

each number is sum of previous two numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 -- --

define 1st two nos as x=0 & y= 1

calculate z= x+y & assign new values of x & y as

```
for (i= 1; i<n ; i++) {
        z= x+y;
        x= y
    }   y= z
```

Imp Note :- The initialisation and update expression of for loop can have multiple expressions seperated by a comma

```
for ( expression 1, expression 2, expression 3; exp. 4 ; expression 5, exp 6 )
{ }
```

5.5.4:- Nesting Loops:

Loop inside body of another loop

```
for ( ____ )
{  for ( ____ ) { }   // for inside for
   while();           // while inside for
}
```

Program:- Armstrong number

Sum of cube of all digits is equal to number

$371 = 3^3 + 7^3 + 1^3 = 371$

```
while (num>0) { rem= num%10;
                result= rem^3
                num /= 10; }
if (num_original == result) { armstrong} else { not armstrong}
```

5.5.6:- Break statement

- used inside loops & switch statement
- used to come out of the loop immediately and execute statement immediately after the loop.

Program:- Program to find prime number

Number is prime if its not divisible by any number for "2 to $\sqrt{n}$"

```
for(i=2; i<=sqrt(n); i++        if (i < sqrt(n))
    if( i%n==0)                     prime number
        break;
```

5.5.7 Continue statement

- used to skip current iteration of the loop & move to next iteration
- break kill all iteration, continue breaks only one iteration

5.5.8 Goto statement

- unconditional statement that forwards program flow to some other part of program
- forward goto: statements between goto & label not executed
- Reverse goto: statements between goto & label is executed repeatedly
- use of goto not favoured in structured programming.

```
goto label;
_ _ _ _ _
_ _ _ _ _
: _ _ _ _
label:
  statements
```

5.5.9 :- Switch statements

- provides choice among number of alternatives
- switch, case & default are keywords
- No need for { } if case has multiple statements
- Switch control exp. is evaluated and control is transferred to lable in case matching with value of switch exp.
- to avoid fall through case "break" is used

(if break not used every thing below case (other cases too) is performed.)

```
switch (expression)
{ case : constant 1
                statement
  case n : constant n
}               statement n
```

5.5.10 :- Pyramids

Type 1

(a)

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

Two for loops : outer for loop for number of lines
inner for loop for no of stars

```
for (i=0; i<5; i++)
    for (j=0; j<i ; j++)
        printf("*");
```

(b)

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

just print value of i in above program

```
for (i=0; i<5; i++)
    for (j=0; j<i ; j++)
        print (i);
```

(c)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

just print value of j

```
for (i=0; i<5; i++)
    for (j=0; j<i ; j++)
        print (j);
```

(d)

```
2
3 4
4 5 6
5 6 7 8
6 7 8 9 10
```

just print value of i+j

```
for (i=0; i<5; i++)
    for (j=0; j<i ; j++)
        print (i+j);
```

(e)

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

print 1 → if (i+j) is even
print 0 if (i+j) is odd

(f)

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

for (f) just take variable m=1 and "print (m++);"

```
int m=1;
for (i=0; i<5; i++)
    for (j=0; j<i ; j<j++)
        print (m++)
```

(g)

```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

print "n+1-j"

(h)

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1
```

print "(n+1-i)"

(i)

Type 2

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

```
for (i=0; i<n; i++)
    for (j=0; j<n-i; j++)
        print (*)
```

logic is simple, we have to print n-i stars on each line

(j)

```
1 1 1 1 1
2 2 2 2
3 3 3
2 2
1
```

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

(k)

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

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

(l)

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

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

(m)

```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
```

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

Type 8

(n)

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

```
for(i=0;i<n;i++){
    for(j=0;j<n-i;j++
        print( ); (space)
    for(j=0;j<i;j++)
        print(*);
```

same as pattern (a)
just add space before *
after (*)

→ note space after *

(o)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
for(i=0;i<n;i++){
    for(j=0;j<n-i;j++
        print( ); (space)
    for(j=0;j<i;j++)
        print(i);
```

→ note space after i

(p)

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

```
for(i=0;i<n;i++){
    for(j=0;j<n-i;j++
        print( ); (space)
    for(j=0;j<i;j++)
        print(*);
```

no space after (*)

(q)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
for(i=0;i<n;i++){
    for(j=0;j<n-i;j++
        print( ); (space)
    for(j=0;j<i;j++)
        print(i);
```

no space after i

(m)

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

```
for(i=1;i<=n;i++)
{ for(j=1;j<n-i;j++)
    print(space)
  for(j=1;j<=2*i-1;j++)
    print(*);
```

then as 2*i-1
star on each
line

(s)

```
1
2 2 2
3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
```

```
for(i=1;i<=n;i++)
{ for(j=1;j<n-i;j++)
    print(space)
  for(j=1;j<=2*i-1;j++)
    print(i)
```

then as 2*i-1
star on each
line

(b)

```
1
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
```

```
for(i=1;i<=n;i++)
{ for(j=1;j<n-i;j++)
    print(space)
  for(j=1;j<=2*i-1;j++)
    print(j)
```

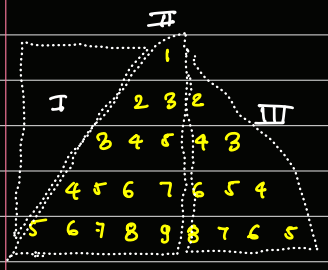
(U)

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

```
for (i=1; i<=n; i++)
{ for(j=1 ; j<=n-i; j++)
    print(space space)        <-- two spaces here
  for(j=1; j<=2*i-1; j++)
    print(*)
```

space

(V)



Imp

we have 2*i-1 no of lines

```
for(i=1;i<=n;i++){
  for(j=1;j<=n-i;j++)        } part I
    print(space)
  for(j=1;j<=i;j++)          } part II
    print(m++);
  m=m-2;                     --> skip one row
  for(j=1;j<=i-1;j++)        } part 3
    print(m--);
```

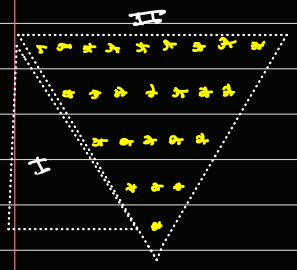
(W)

Imp

```
m=n
for(i=1;i<=n;i++){
  for(j=1;j<=n-i;j++)        } part I
    print(space)
  for(j=1;j<=i;j++)          } part II
    print(m--)
  m=m+2;
  for(j=1;j<=i-1;j++)        } part 3
    print(m++);
```

```
5
5 4 5
5 4 3 4 5
5 4 3 2 3 4 5
5 4 3 2 1 2 3 4 5
```

(X)




```
for(i=1;i<=n;i++)
  for(j=1;j<=i;j++)          } part I
    print(space)
  for(j=1;j<=(2*(n-i)+1);j++) } part II
    print(*)
```

for each line i, there are 2*(n-i)+1 stars to print

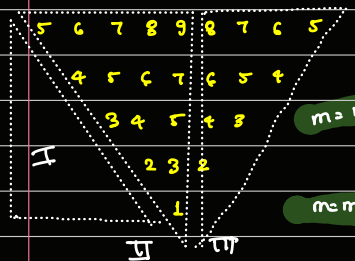
(Y)

```
5 5 5 5 5 5 5 5 5
4 4 4 4 4 4 4
3 3 3 3 3
2 2 2
1
```

```
for(i=1;i<=n;i++)
  for(j=1;j<=i;j++)          } part I
    print(space)
  for(j=1;j<=(2*(n-i)+1);j++) } part II
    print(n-i+1)
```

Imp

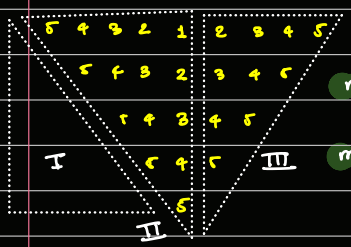
(21)




```
for (i=1; i<n; i++)
    for (int j=0; j<=i; j++)
        print (space)
m = n+i-1 → for (j=1; j<=(n-i+1); j++)
        print (m++)
m = m-2 → for (j=1; j<=n-i; j++)
        print (m--);
```

Imp

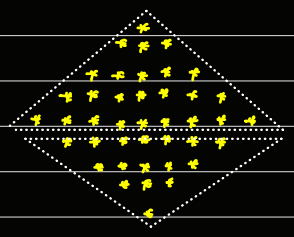
(22)




```
for (i=1; i<n; i++) {
    for (j=1; j<=i; j++)
        print (space);
m=n → for (j=1; j<=(n-i+1); j++)
        print (m++);
m=m-2 → for (j=1; j<=n-i); j++)
        print (m--)
```

Complete Problems on pattern :-

(23)



```
for (int i=1; i<=n; i++) {
{   for (j=1; j<=i; j++)
        print (space);
    for (j=1; j<=2*i-1; j++)
        print (*)
}
for (int i=1; i<=n; i++) {
{   for (j=1; j<=i; j++)
        print (space)
    for (j=2; j<=2*n-1; j++)
        print (*)
}
```

upper △

lower ▽

Notice the difference