CS & IT ENGINEERING

Control flow statements

Switch Statement

Lecture No. 04

By- Pankaj Sharma sir

1-Switch statement

```
for ( i=1; i<=n; i=i×3)
    {
        for(j=i; j<=n ; j++)
            {
            printf ("Pankaj");
            }
    }
```

dependent loop

$i=1$	$i=3^1$	$i=3^2$	$i=3^k$
$j=1,2,3,\ldots n$	$j=3,4,5,\ldots n$	$j=3^2,10,11,\ldots n$	$j=3^k$ to n
1 to n	3 to n	3^2 to n	$(n-3^k+1)$
$(n-1+1)$	$(n-3+1)$	$(n-3^2+1)$	

How many terms?

$$\underbrace{(n-1+1)}_{1}+\underbrace{(n-3^1+1)+(n-3^2+1)+\cdots+(n-3^k+1)}_{k}$$

$$(k+1)(n+1) - 1-3^1-3^2\cdots-3^k$$

$$(k+1)(n+1) - (1+3^1+3^2+\cdots+3^k)$$

$$(k+1)(n+1) - \frac{3^{k+1}-1}{3-1}$$

$$(k+1)(n+1) - \frac{3^{k+1}-1}{2}$$

$$3^k <= n$$

$$k <= \log_3 n$$

$$k = \lfloor \log_3 n \rfloor$$

$$\left(1+\lfloor \log_3 n \rfloor\right)(n+1) - \frac{1}{2}\left(3^{1+\lfloor \log_3 n \rfloor}-1\right)$$

2+3X5 → Evaluate → 17

```
switch(expression/condition){
    case constant1 :
        block of statements
        break;
    case constant2 :
        block of statements
        break
    case constant3 :
        block of statements
        break
    default :
        block of statements
        break;
}
```

Switch statement

* Keyword : used to create selection statement with multiple choice.

* Multiple choices are provided with another keyword : case

```
switch(n){

    case1:   Code we want to execute if the value of
             n is 1
             break;
    case2:   Code we want to execute if the value of
             n is 2
             break;

    default: Code we want to execute if the value
             of n does not match any case label.
             break;

}
```

```
int i = 3;
switch(i){
```

1st step : Matching

```
case 1 :

case 3 :

default :

}
```

```
printf("One");
break;

printf("Three");
break;

printf("Wrong");
break;
```

Sequential

Three

Falling through case

```
int i=3;
switch(i) {
```

1st step: Matching

```
case 3 :
    printf("Three");

case 4 :
    printf("Four");
    break;

default :
    printf("Wrong");
    break;
}
```

Three Four

1. break is optional
2. exp → evaluate to a int value.
3. position of $\boxed{\text{default}}$ does not matter, it can be anywhere → start, end, ...
4. default is optional
5. Duplicate case labels are not allowed.
6. Case label can not be a variable.
7. range ⇒ case low_value ... highvalue
   - 1 space
   - (Not on all compiler)
8. set of values ⇒ same code

```
switch(i){
case 'A':
Error
Case 65:
```

'A' → 0101 1112

65 → shaktiman

```
switch(i){
    case 3 :
        break;
    case 4 :
        break
    default :
        break;
}
```

```
int i = 3
switch(i){
    case 3 :
        break
    case 1 :
        break
    default :
        break
}
```

int value

```
switch(expression)
{
}
```

Expression
- 12
- 1+2x3
- !2
- 12 & 3
- 12 && 30
- printf("Pankaj")
- a+bx2

int a=1,b=2;

Valid

switch(12.5)

float

{ }

Ud ke round kick

```
switch('A'){
    case 65 :  [code]
               break;
    case 70 :  [code]
               break
    default :  [code]
               break
}
```

'A' == 65 ⇒ valid

'A' == 70 ⇒ valid

'A' + 2 valid expression

↓

65 + 2

```
switch('A'+2)
{
  ==
}
```

```
int i=3;
switch(i+3){
```

6

```
    case 5 : printf("5");
             break;

    default : printf("0")
              break;

    Case 7 : printf("7");
             break;
}
```

Valid

```
int i=3;
switch(i+3){
        case 1 : printf("1");
                 break;

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

    }
```

6

Valid

```
int i=3;
switch(i+3){

}
```

dummy

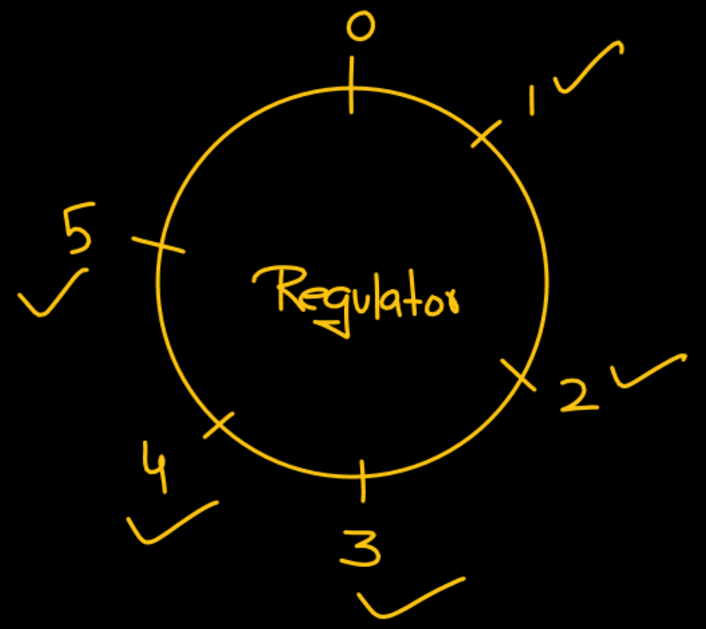
Valid

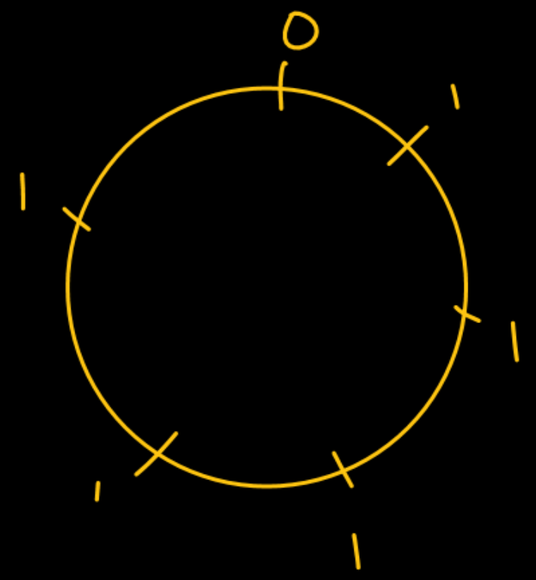
```
switch(i+3);
```

valid

```
switch( );
```

```
switch(i){

  case 4: ____
          break;

  case 2+2: ____
            break;
}
```





No duplicate case labels

```
int i=3,a=10,b=20;
switch(i+3)
{
case a :
    break;
case b :
    break;
case 10 :
    break;
}
```

case labels ⇒ literals

case 10 : ✓

OK

case a : ✗ → Invalid

case 10+3×4 :

OK

```
switch(i+3){
    case 1 :  printf("1");
              printf("2");
              break;

    case 2 :  printf("3");
              printf("4");
              break ;

}
```

```
switch(i+3){
    case 1 :  printf("1");
              printf("One");
              break;
              printf("Pankaj");

    case 2 :  printf("2");
              printf("Two");
              break;

}
```

Never gets printed

No Error

```
int i=2;
switch(i) {
        i= i+2;
  case 2 : printf("2");
           break;
  case 4 : printf("4");
           break;
        }
printf("%d", i);
```

ignore

22

Range

```
if ( n>=1 && n<=60)
{

}
else {

}
```

Valid

Not for all compilers

1 → 10

```
switch(i){

case 1 ... 10 :
    break;

}
```

low high

space

case 10 ... 1 : ✗

Error

Set of values → Code

```
if ( n==1 || n==13 || n==18)
{

}
```

Valid

```
else if ( n==2 || n==10 || n==15)
{

}
```

```
switch(i){

case 1 :
case 13 :
case 18 :  print("1");
           break;

}
```

i=1

i

$\boxed{\cancel{0}\,2}$

```
switch(i=2){
    i= i+4;
}
printf("%d", i);
```

```
for (i=1; i<=10; i++)
    {

        printf("%d",i);
    }
```

O/P: 1 2 .. _ 10

continue

skip the remaining code of current iteration and continue with next iteration

```
for (i=1; i<=10; i++)
    {

        if (i % 3 == 0)
            continue;

        printf("%d",i);
    }
```

i ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7

1 % 3 == 0 ⇒ False
2 % 3 == 0 ⇒ False
3 % 3 == 0 ⇒ True
4 % 3 == 0 ⇒ False
5 % 3 == 0 ⇒ False
6 % 3 == 0 ⇒ True

continue

O/P: 1 2 4 5 7 8 10

$1 \times 2 \times 3 \times 4 \times 5$

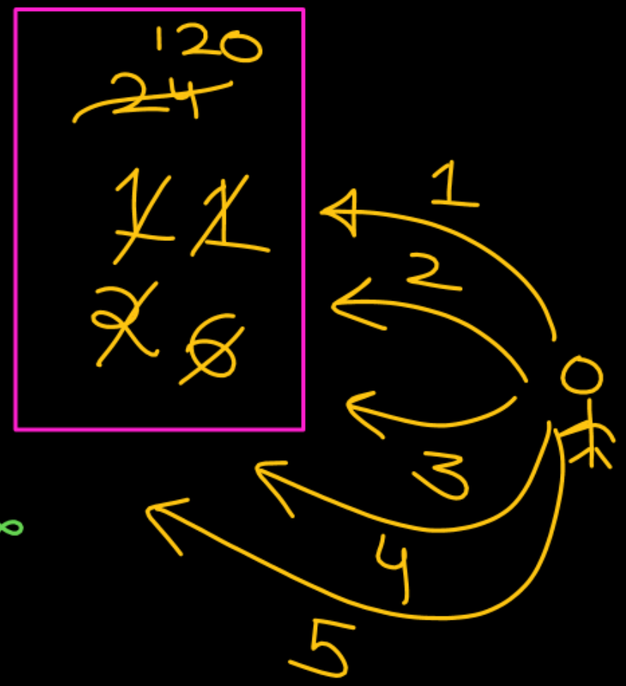
```
AbtakkaProd = 1;

AbtakkaProd = AbtakkaProd x 1;
AbtakkaProd = AbtakkaProd x 2;
AbtakkaProd = AbtakkaProd x 3;
AbtakkaProd = AbtakkaProd x 4;
AbtakkaProd = AbtakkaProd x 5;
```

vary variable

Repeat

```
AbtakkaProd = 1;
for (i=1; i<=5; i++)
    AbtakkaProd = AbtakkaProd x i;
```

AbtakkaProd



```
Prod = 1 ;
for ( i=1; i<=5; i++)
        Prod = Prod x i ;
printf("%d", Prod);
```

5!

```
Prod = 1
for ( i=1; i<= 10; i++)
        Prod = Prod x i;
printf("%d", Prod);
```

10!

n! ⟸

```
Prod = 1;
for( i=1; i<= n; i++)
        Prod = Prod x i;
printf ("%d", Prod);
```

\n ⟹ ✓

\t ⟹ tab


THANK YOU
GW
SOLDIERS !