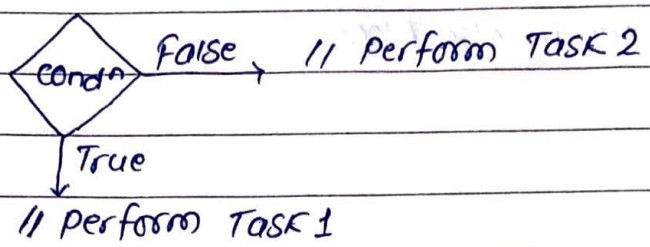
Conditions & Loops

Date: .....
Page: .....

→ Conditional & Decision Making statements.

Conditional statements are used to check whether the given condition is true or false. Do the given task1, if the condition is true & if the condition is false execute the task2.



→ Decision Making statements.

1) If Statement.

Syntax :
```
if (condn) {
    // statements
}
```

working of if statement

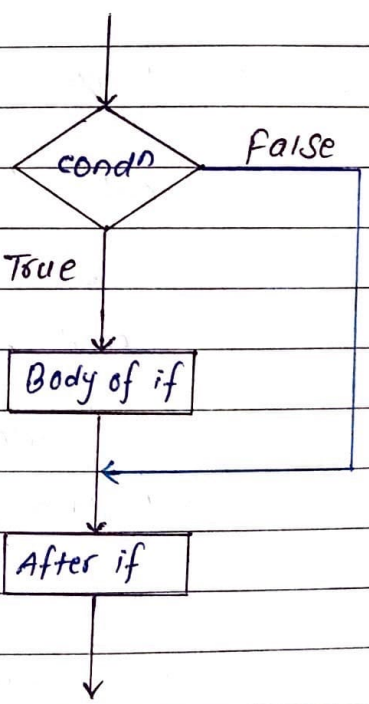
i) condition is True

```
int num = 10;
if (num > 0) {
    // code
}
// code after if
```

ii) condition is false

```
int num = 10;
if (num < 0) {
    // code
}
// code after if
```

flow chart of if statement



Date: .....
Page: .....

2) If ... else statement :-

Syntax :

```
if (condn) {
    // code in if block
}
else {
    // code in else block
}
```

working of if ... else statement

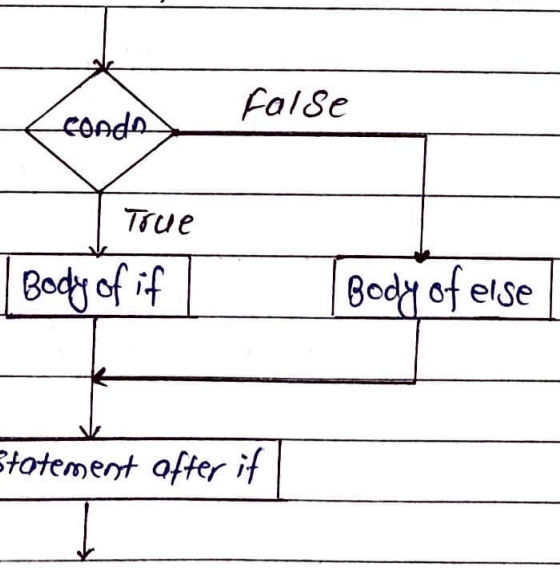
i) condition is true

```
int num = 5;
if (num > 0) {
    // code
}
else {
    // code
}
// code after if ... else
```

ii) condition is false

```
int num = 5;
if (num < 0)
    // code
}
else {
    // code
}
// code after if ... else
```

flow chart of if ... else statement



Date: .....
Page: .....

3) if --- else --- if statement

Syntax :

```
if (condn1) {
    // if condn 1 is true, execute this block
}
else if (condn 2) {
    // Execute this block if condn 1 is false & condn 2 is True.
}
else if (condn 3) {
    // Execute this block if condn 1 & 2 are false & condn 3 is True.
}
.
.
.
else {
    // if none of the condn is true, than this block execute.
}
```

Working of if -- else -- if statements.

i) 1st condn is true

```
int num = 2;
if (num > 0) {
    // code
}
else if (num == 0) {
    // code
}
else {
    // code
}
// code after
if -- else -- if
```

ii) 2nd condn is true

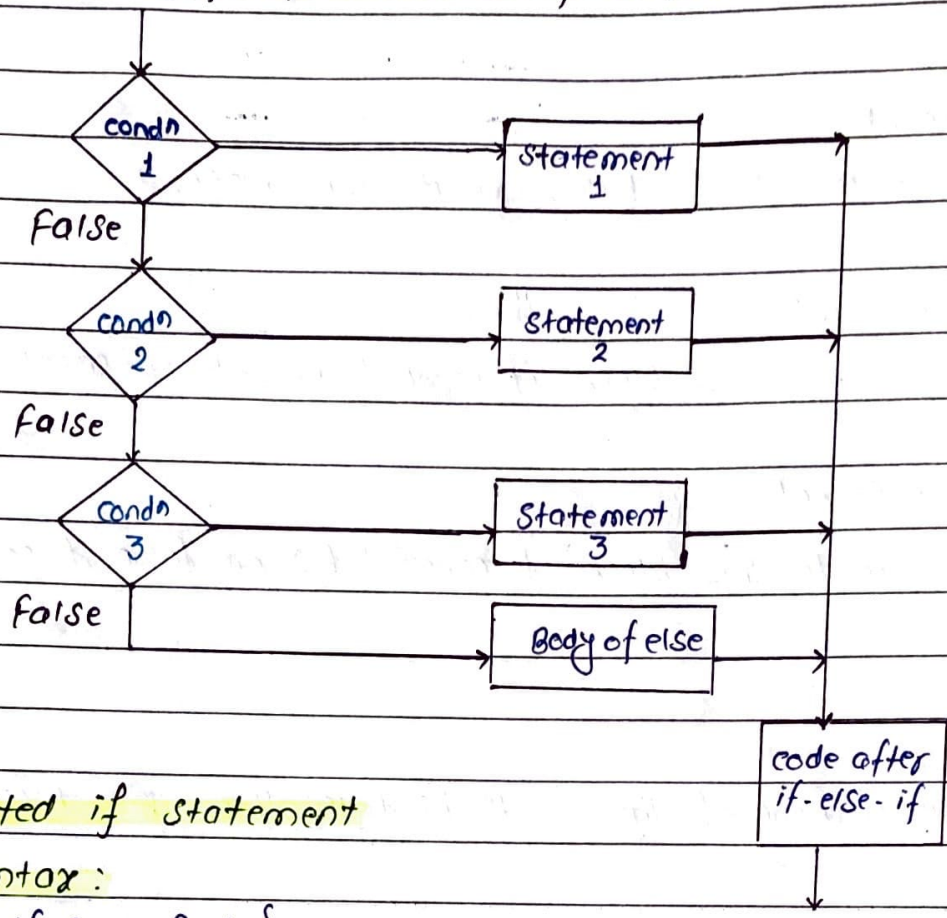
```
int num = 0;
if (num > 0) {
    // code
}
else if (num == 0) {
    // code
}
else {
    // code
}
// code after
if -- else -- if
```

iii) All condn are false

```
int num = -2;
if (num > 0);
    // code
}
else if (num == 0) {
    // code
}
else {
    // code
}
// code after
if -- else -- if
```

Date: .....
Page: .....

Flow chart of if---else---if statement



4) Nested if statement

Syntax:

```
if (condn 1) {
    // when condn 1 true
    if (condn 2) {
        // when condn 2 true
    }
}
else {
    // code
}
```

NOTE :- If we execute a single line after if condn, then no need to use curly braces. you can use semicolon (;) But if you execute multiple line after if condn, you have to use curly braces.

Date: .....
Page: .....

→ **LOOPS :-**

There is need to perform some operation more than once. Then, loops come into use. When we need to repeatedly execute a block of statements.

There are mainly Two Types of Loop.

i) Entry controlled
   - for loop
   - while loop

ii) Exit controlled
   - do-while loop.

→ **Entry controlled Loops.**

The test condⁿ is tested before entering the body of loop. for & while loop are entry controlled loops.

→ **Exit controlled Loops.**

The test condⁿ is tested at the end of the loop body. Therefore, the loop body will execute atleast once irrespective of whether the condⁿ is true or false. The do-while loop is exit controlled loop.

1) **For Loop**

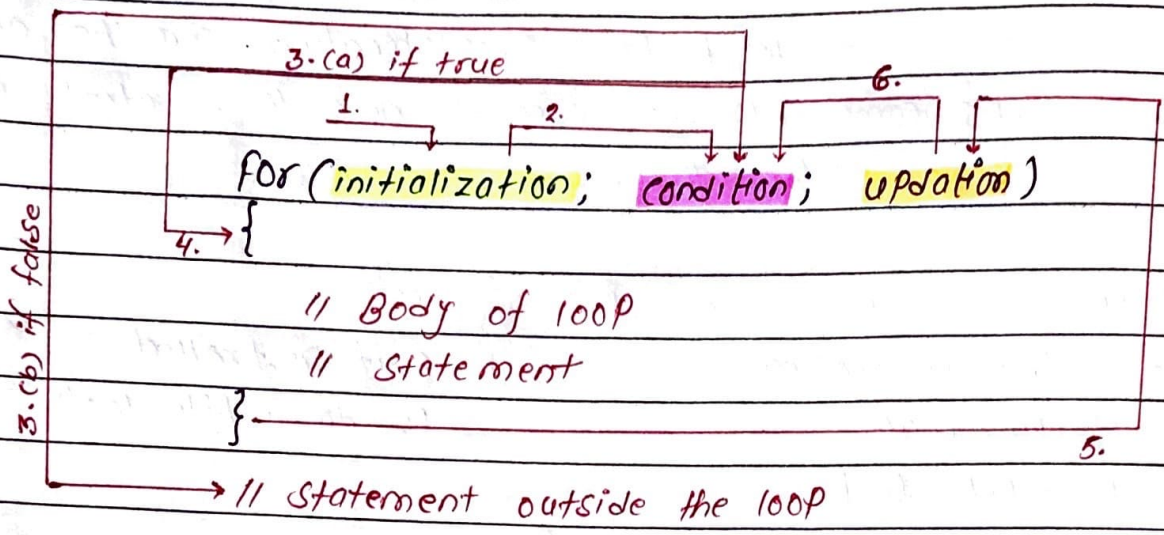
When you know exactly how many times you want to loop through a block of code, use the for loop.

Syntax :

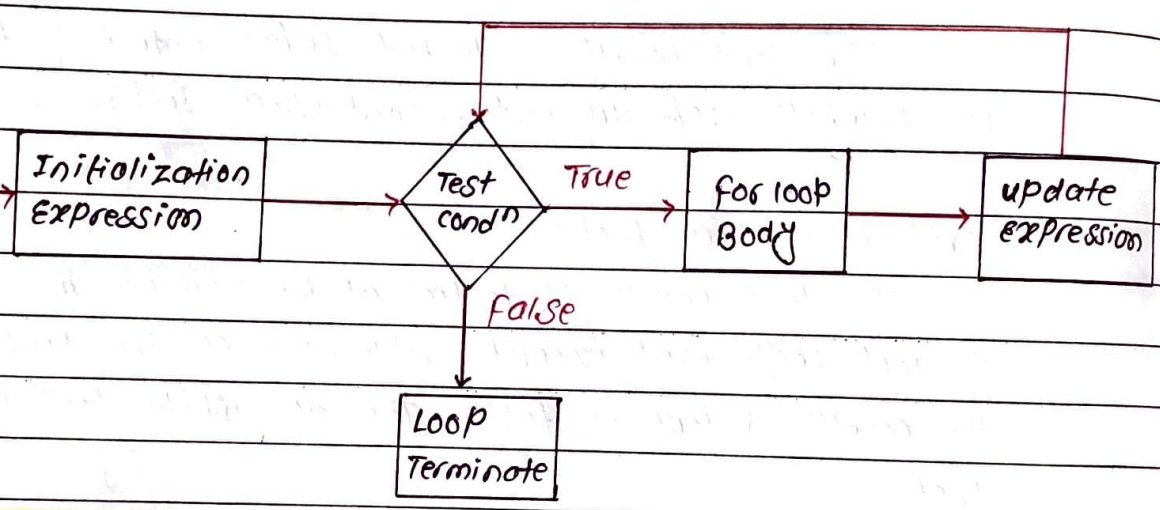
```
for (initialization; condition; updation) {
    // Body of loop
    // statements
}
```

Date: .....
Page: .....

How for Loop works.



Flow Chart:



Infinite for Loop

A loop that never stop executing.

Syntax:
```
for (;;) {
    // code
}
```

eg:
```
int main () {
    for (int i = 1; i <= 10; i--) {
        cout << "Hello" << endl;
    }
}
```

Date: .....
Page: .....

→ **For-Each Loop:**

It is used execusively to loop through element in an "array".

Syntax:

```
for(Data_Type  Variable_name  :  arrayName) {
        // code
}
```

Eg:-
```
String[] cars = {"BMW", "Toyota", "Fords", "Benz"};
for (string car : cars) {
        cout << car << endl;
}
```

→ **Nested For Loop:**

Nested loop means, a loop statement inside another loop statement

Syntax:

```
for (initialization ; condition ; updation) {
    for (initialization ; condition ; updation) {
        // statement of inner loop
    }
    // statement of outer loop
}
```

2) **While Loop**

The while loop is considered as a repeating if statement. If the no. of iteration is not fixed, it is recommended to use the while loop.

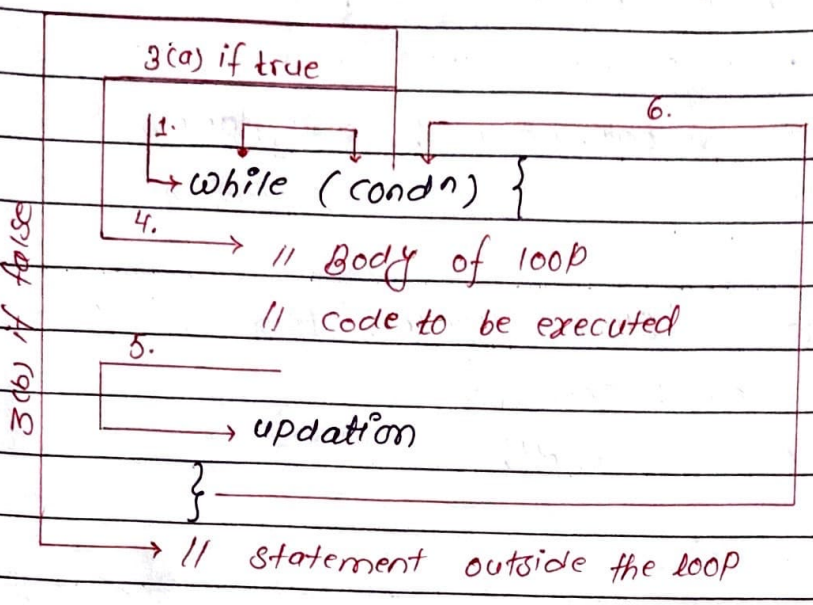
Syntax:

```
while (condn) {
    // code
    updation (Increment / Decrement)
}
```

Date: .....

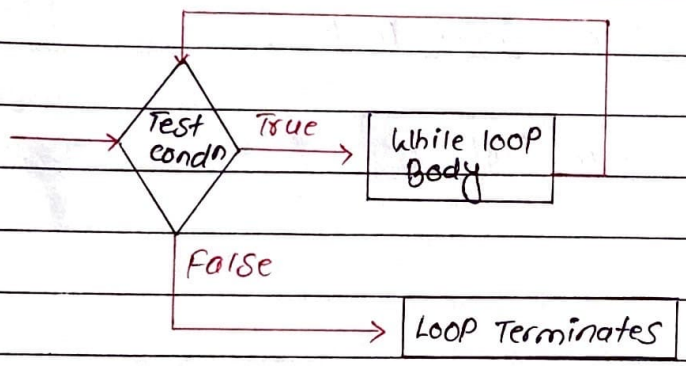
Page: .....

→ **How while Loop works:**



NOTE: DO not forget to update the variable used in the condition, otherwise the loop will never end!

→ **Flow chart:**



→ **Infinite while Loop:**

Syntax:

```
white (true) {
    // code to executed
}
```

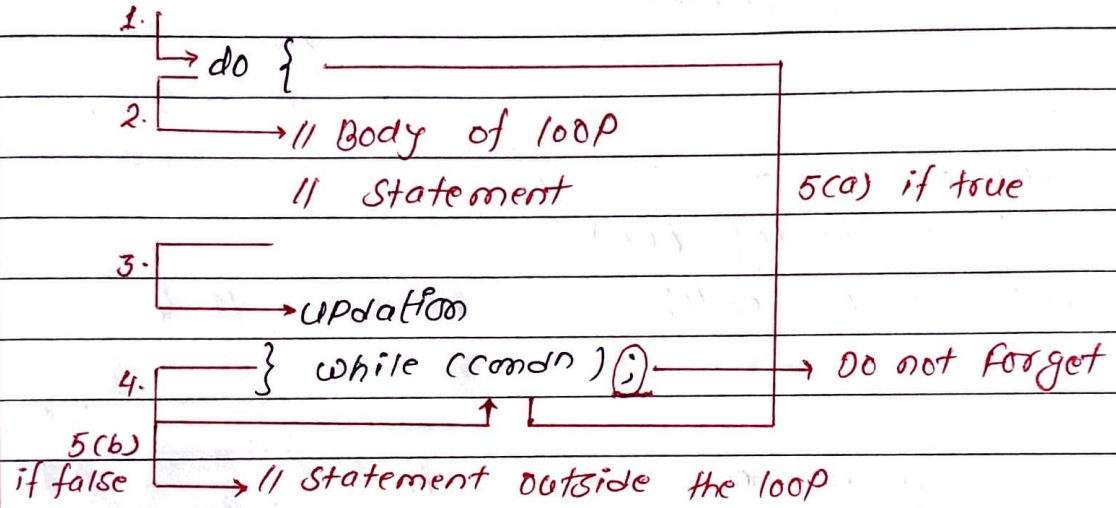
Date: .....
Page: .....

3) **do-while LOOP**

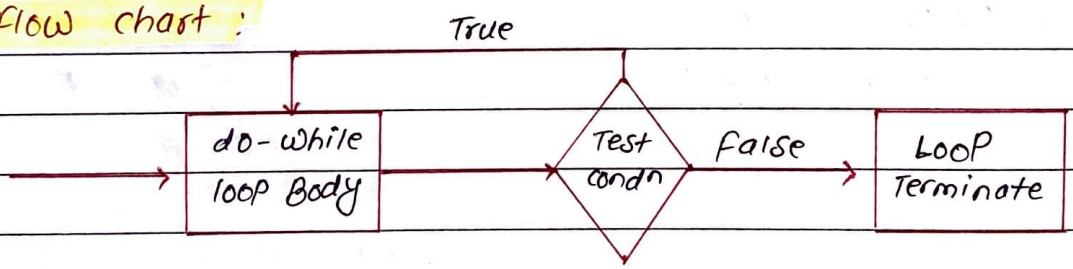
If the no. of iteration is not fixed & you must have to execute the loop atleast once, it is recommended to use a do-while loop.

Syntax:
```
do {
    // Body of loop
    updation
} while (condn);
```

→ **How do-while loop works:**



→ **Flow chart:**



→ **Infinite do-while loop**

```
do {
    // code
} while (true);
```

Date: .....

Page: .....

→ Example of for loop:

```
for (int i = 0; i<3; i++) {
    cout << " outer loop" << i << endl;
    for (int j = 0; j<3; j++) {
        cout << "Inner loop" << j << endl;
    }
}
```

→ Print your Name 11 Times

```
for (int i=1; i<=11; i++) {
    cout << " Suraj" << endl;
}
```

→ Print Table of 19

```
for (int i = 1; i<=10; i++) {
    cout << 19*i << endl;
}
```

→ Print only Even Numbers from 1 - 100

```
for (int i = 1; i<=100; i++) {
    if (i % 2 == 0) {
        cout << i << " ";
    }
}
```

Date: .....
Page: .....

→ Break & continue.

1) Break -

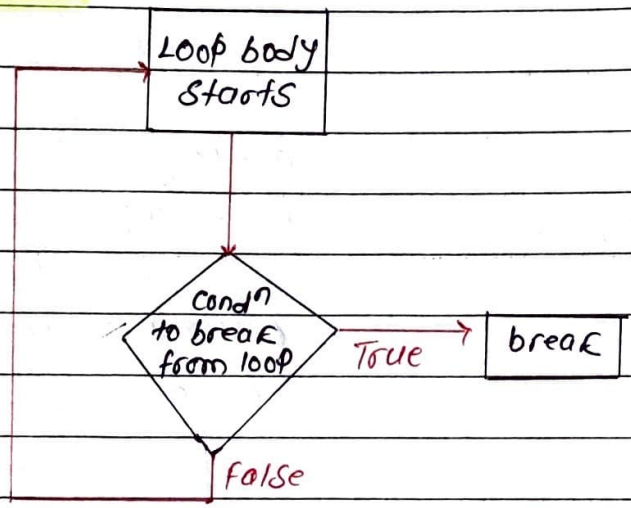
The break statement is usually used to come out of the loop instantly. Whenever a break statement is encountered inside a loop, the control directly comes out of loop & the loop gets terminated for rest of the iterations.

Break = To exit from the loop.

NOTE: When a break statement is used inside a nested loop then only the inner loop gets terminated.

Syntax: break;

Flow chart:



How break statements works?

```
while (condn) {
    // codes
    if (condn to break)
        break;
    // codes
}
```

```
do {
    // codes
    if (condn to break)
        break;
    // codes
} while (condn);
```

Date: .....
Page: .....

```
for (initialization ; condition ; updation) {
    // codes
    if (condn to break)
        break;
    // codes
}
```

```
int main () {
    for (int i = 0; i<5; i++) {
        cout << i << endl;
        if (i==3)
            break;
    }
}
```

output = 0
1
2
3

```
int main () {
    for (int i = 0; i<=5; i++) {
        cout << "Hello" << endl;
        if (i==4)
            break;
    }
}
```

output: Hello
Hello
Hello
Hello
Hello

```
int main () {
    for (int i = 100; i>=10; i--) {
        if (i==99)
            break;
    }
    cout << "out of loop";
}
```

output:
out of loop

Date: .....
Page: .....

2) **Continue -**

The continue statement is used when we want to skip a particular condⁿ & continue the rest execution.

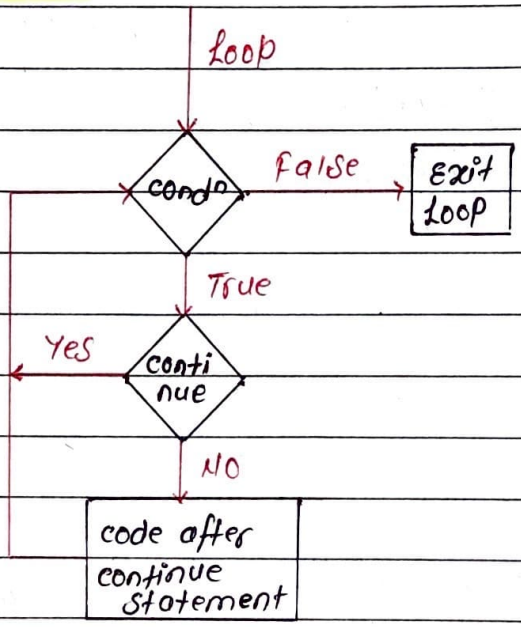
continue : To skip the iteration.

Incase of for loop, the continue keyword force control to jump immediately to the update statement.
Incase of while or do-while loop, control immediately jumps to the Boolean expression.

NOTE : The continue statement is almost always used in decision making statements (if... else statement)

Syntax : continue;

Flow chart :



Date: .....
Page: .....

Working of continue Statement

```
while (condn) {
    // codes
    if (condn)
        continue;
    // codes
}
```

```
do {
    // codes
    if (condn)
        continue;
    // codes
} while (condn);
```

```
for (initialization; condition; updation) {
    // codes
    if (condn)
        continue;
    // codes
}
```

```
int main() {
    for (int i = 1; i <= 10; i++) {
        if (i > 4 && i < 9) {
            continue;
        }
        cout << " " << i;
    }
}
```

output : 1 2 3 4 9 10

```
int main() {
    for (int i = 0; i <= 5; i++) {
        cout << "Hello" << endl;
        continue;
        cout << "Good Morning";
    }
}
```

output :
Hello
Hello
Hello
Hello
Hello
Hello

Switch Case in C++

Date: .....
Page: .....

+ **Switch case** - It is used when we have no. of options (choice) & we may need to perform a different task for each choice.
+ It is an alternative of long if-else statement.
+ Expression must have an integer value.

+ **Syntax of switch case:**

```
switch (expression) {
    case value 1 :
        // statement;
        break;
    case value 2 :
        // statement;
        break;
    ------
    ------
    ------
    default :
        // default statements;
        break;
}
```

+ **How does the switch case works?**

The expression is evaluated once & compared with the value of each case.

1) If expression matches with value 1, the code of case value1 are executed. Similarly, the code of case value2 is executed if expression matches with value 2.

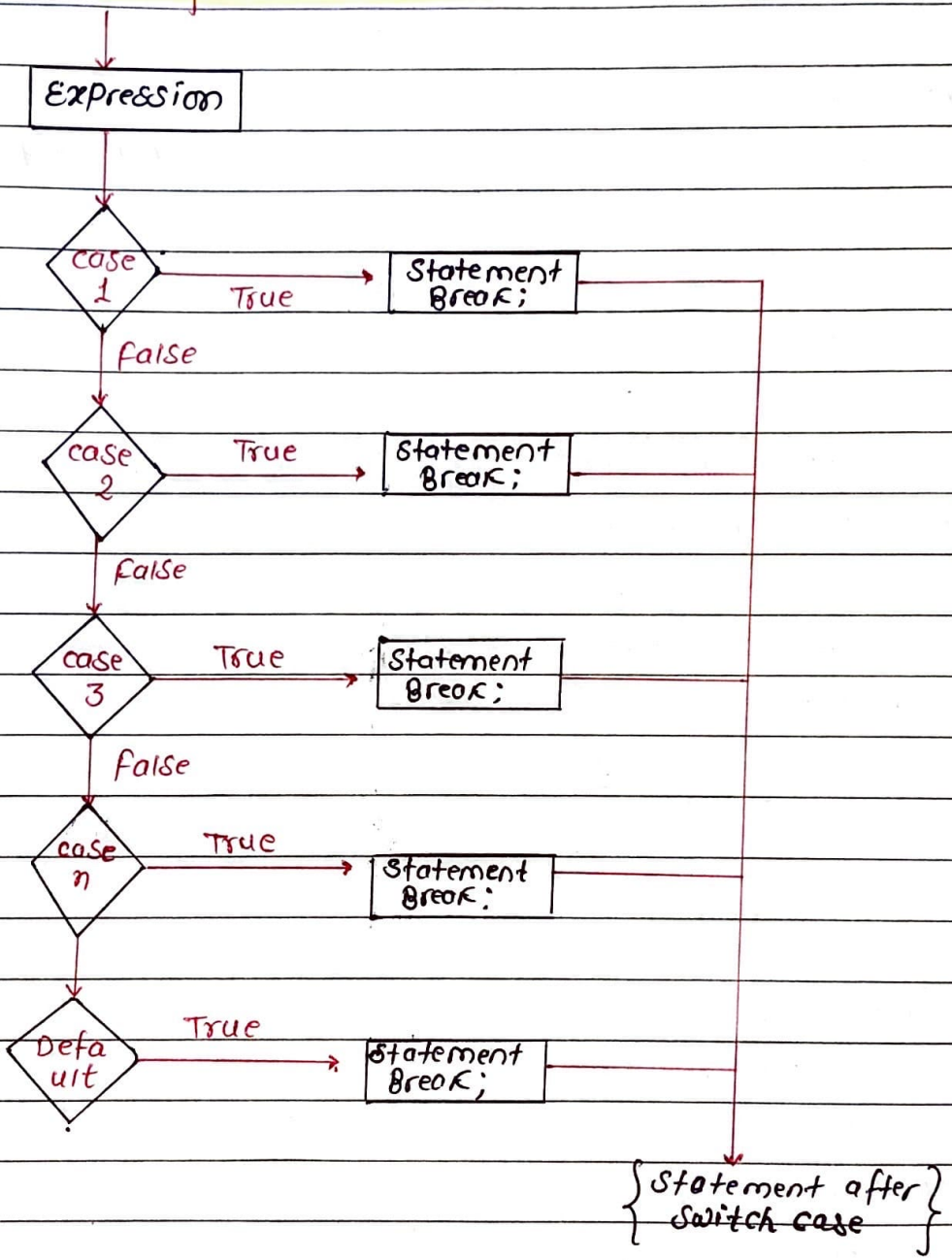
2) If there is no match, the code of default case is executed.

Date: .....
Page: .....

→ **Why didn't use break statement after default ?**

The control would itself come out of the switch after default, so we didn't use it. However if you still want to use the break after default then you can use it.

→ **Flow chart of Switch case**



Date: .....
Page: .....

→ **Rules of the switch case:**

1) The case value must be either int or char type.

```
switch (1+2+3)
"    (1*2%4)
char ch = 'b';
switch (ch)
```
} Valid expression (constant value)

```
switch (a+b+c)
"    (ab+cd)
```
} Invalid expression (Doesn't result in a constant value).

2) There can be any no. of cases.

3) No duplicate case values are allowed

4) Each statement of the case can have a break statement. It is optional.

5) The default statement is also optional.

→ **Simple calculator using the switch case.**

```
int main() {
    char choice;  int x, y;  int ans;
    cout << " Enter 2 numbers : " << endl;
    cin >> x >> y;
    cout << " Enter the operator (+,-,*,/) : " << endl;
    cin >> choice;
    switch (choice) {
        case '+' :
            ans = x+y;
            break;
        case '-' :
            ans = x-y;
            break;
        case '*' :
            ans = x*y;
            break;
        case '/' :
            ans = x/y;
```

Date: .....
Page: .....

```
        break;
    default:
        cout << " Invalid operator." << endl;
    }
    return 0;
}
```

```
int main() {
    char ch = 'b';
    switch (ch) {
      case 'd':
          cout << " case 1" ;
          break;
      case 'b':
          cout << " case 2" ;
          break;
      case 'x':
          cout << " case 3" ;
          break;
      case 'y':
          cout << " caseA" ;
          break;
      default;
          cout << " Default case" ;
    }
    return 0;
}
```

Ternary operator :-

It is the only operator that takes "three" operands. This operator is frequently used as a one line replacement for if--- else statement.

Syntax :

variable = condn ? expression 1 :
expression 2 ;

If condn is true, expression 1 is executes.
If condn is false, expression 2 is executes.

Eg:- int num = 4;

```
string type = (num %02 == 0) ?
                "even" : "odd" ;
 cout << type ;
```

output : even