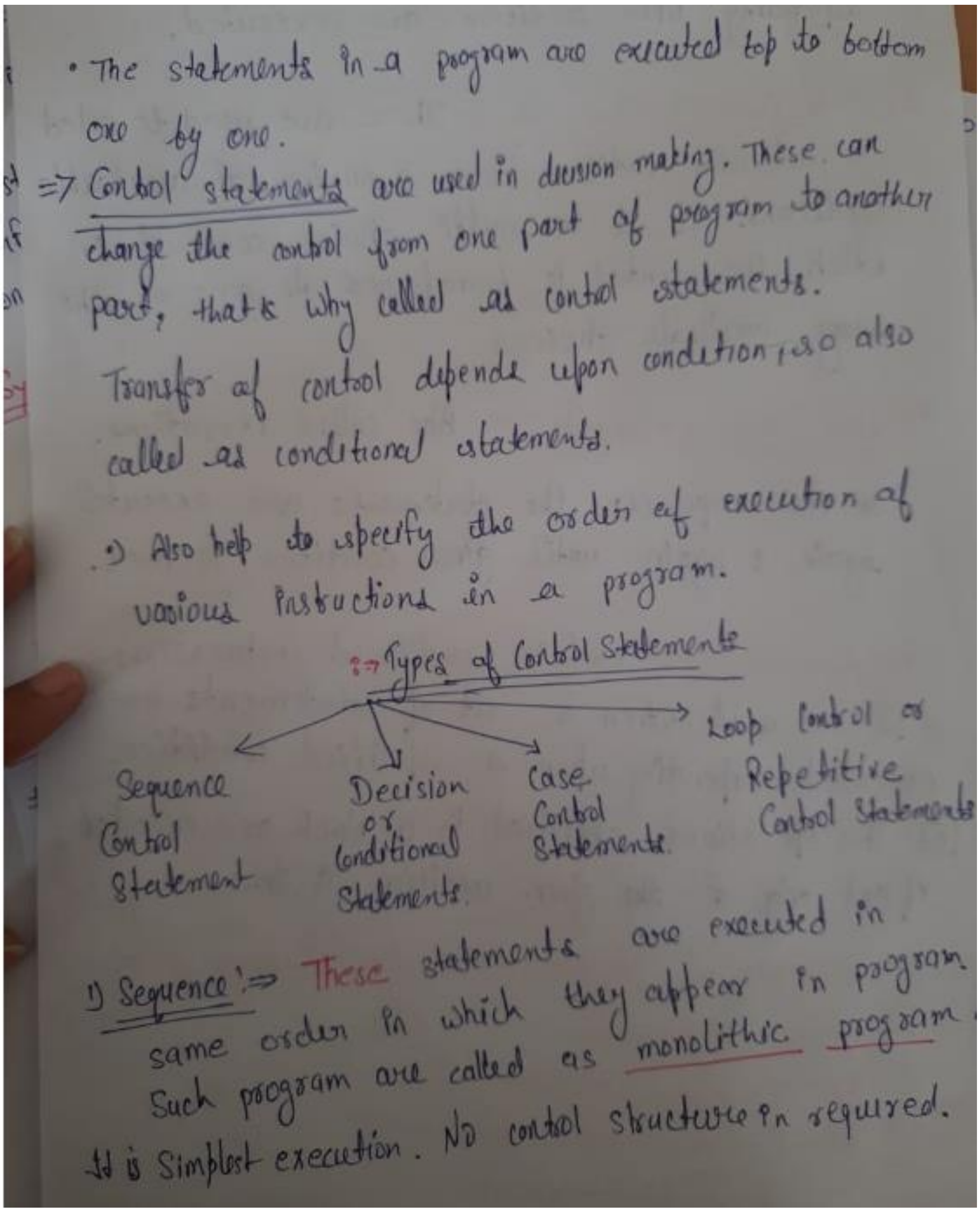
Unit-3(Control Structure)



• The statements in a program are executed top to bottom one by one.

=> Control statements are used in decision making. These can change the control from one part of program to another part, that is why called as control statements. Transfer of control depends upon condition, so also called as conditional statements.

•) Also help to specify the order of execution of various instructions in a program.

=> Types of Control Statements

$$\begin{tikzcd}
 & & \text{Types of Control Statements} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Sequence Control Statement} & \text{Decision or Conditional Statements} & \text{Case Control Statements} & \text{Loop Control or Repetitive Control Statements}
\end{tikzcd}$$

1) Sequence :=> These statements are executed in same order in which they appear in program. Such program are called as monolithic program. It is Simplest execution. No control structure is required.

2) Decision Control Structure:-> Sequence of statements depending upon condition are executed. ②

(3) Case Control Statements:=> These are used to select a particular condition from a number of available conditions. It is a multi choice construct in which the control is transferred to one of the many available choices.

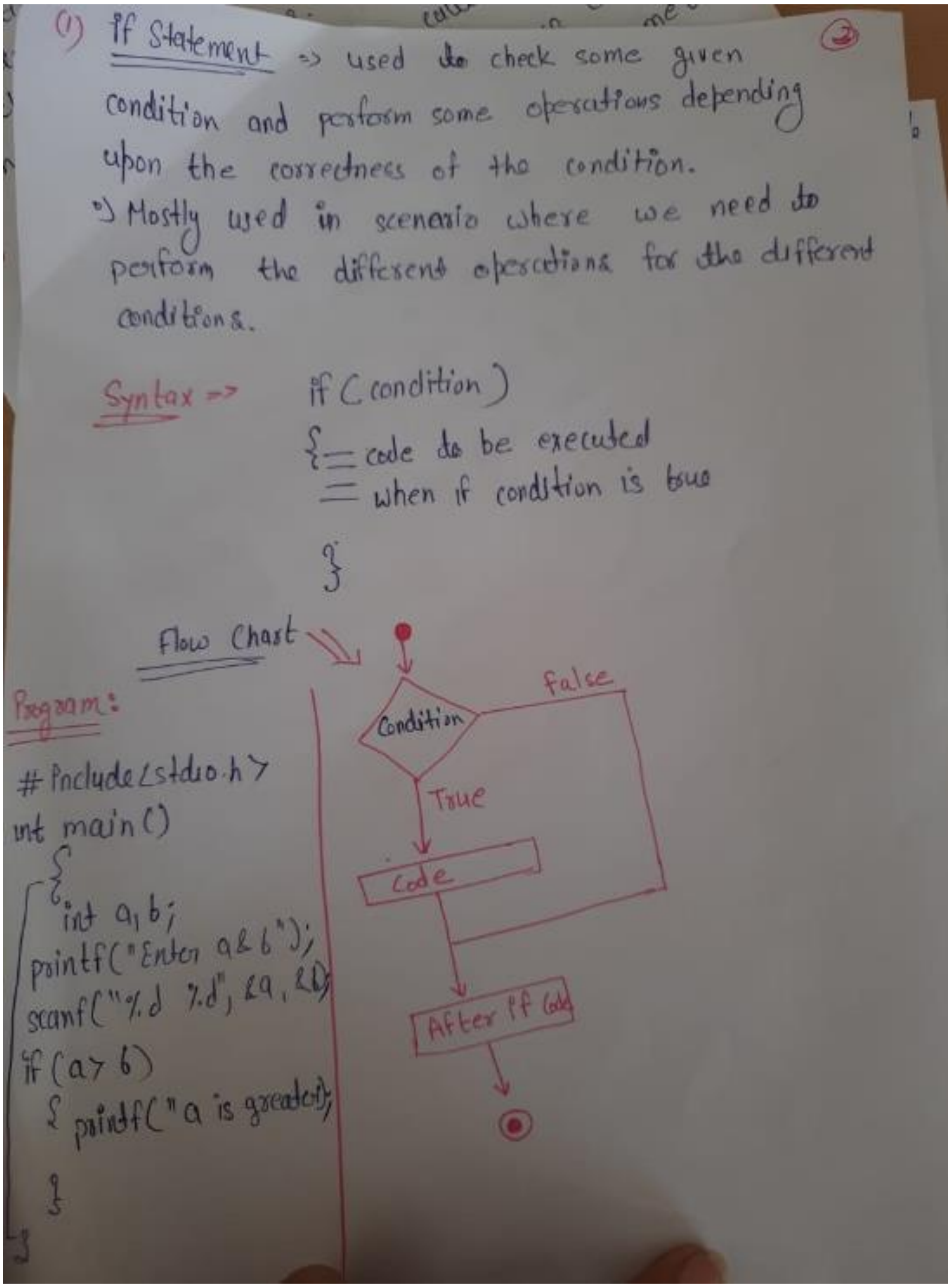
(4) Loop Control Statements:-> Also called iterations.

In this sequence the statements are executed again & again until given condition is met.

If statements => Used in conditional expressions.
•) It is used when a set of statements are executed depending upon a specified condition.
(•) The operations specified in if block are executed if and only if the given condition is true.

Variants of If statements in C-Language:->

(1) If -Statement
(2) if -else Statement
(3) if-else-if ladder Statement
(4) Nested if Statement.



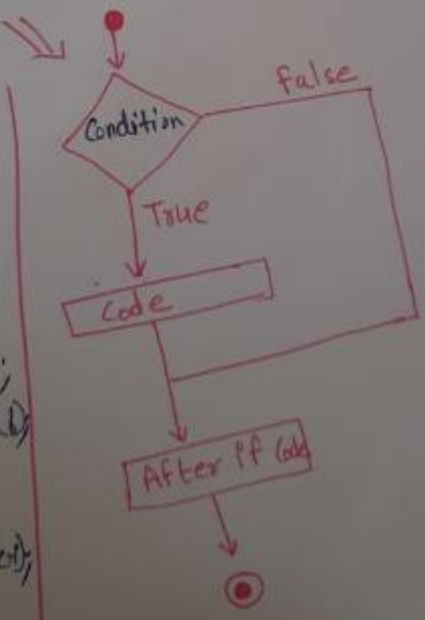
(1) If Statement => used to check some given condition and perform some operations depending upon the correctness of the condition.

*) Mostly used in scenario where we need to perform the different operations for the different conditions.

②

Syntax =>
```
if (condition)
{ = code to be executed
  = when if condition is true
}
```

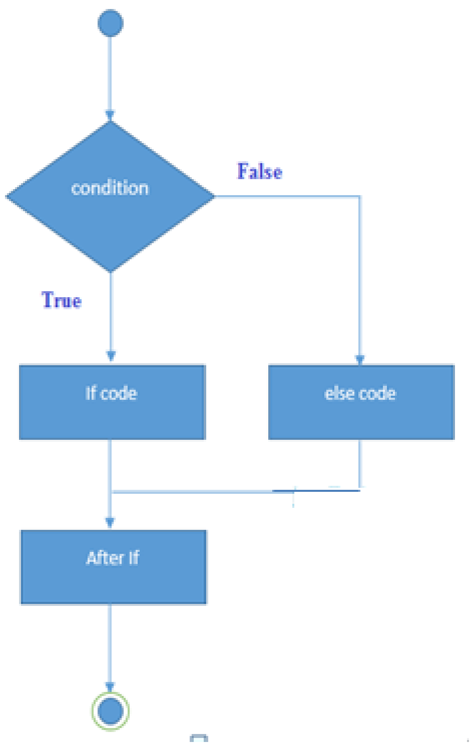
Flow Chart



Program:
```
#include<stdio.h>
int main()
{
 int a,b;
printf("Enter a&b");
scanf("%d %d", &a, &b);
if (a>b)
 { printf("a is greater");
 }
}
```

2) if-else:→It is used if there are two possible results of a questions. It **executes the if *block* if condition is true otherwise *else block* is executed.**

Syntax:➔
```
if(condition){
                    //code if condition is true
                  }
 else         {
                    //code if condition is false
                    }
```



```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a,b;
  clrscr();

  printf("\n Enter any Two numbers\n");

  printf("\n Enter First Number : ");
  scanf("%d",&a);

  printf("\n Enter Second Number : ");
  scanf("%d",&b);

  if(a>b)//logical test
   {
    printf("\n %d is Greater",a);
   }

  else
   {
    printf("\n %d is Greater",b);
   }

 getch();
}
```

3) if-elseif-.....else Ladder:→ **The if-else-if ladder statement executes one condition from multiple statements**.

Syntax:→

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

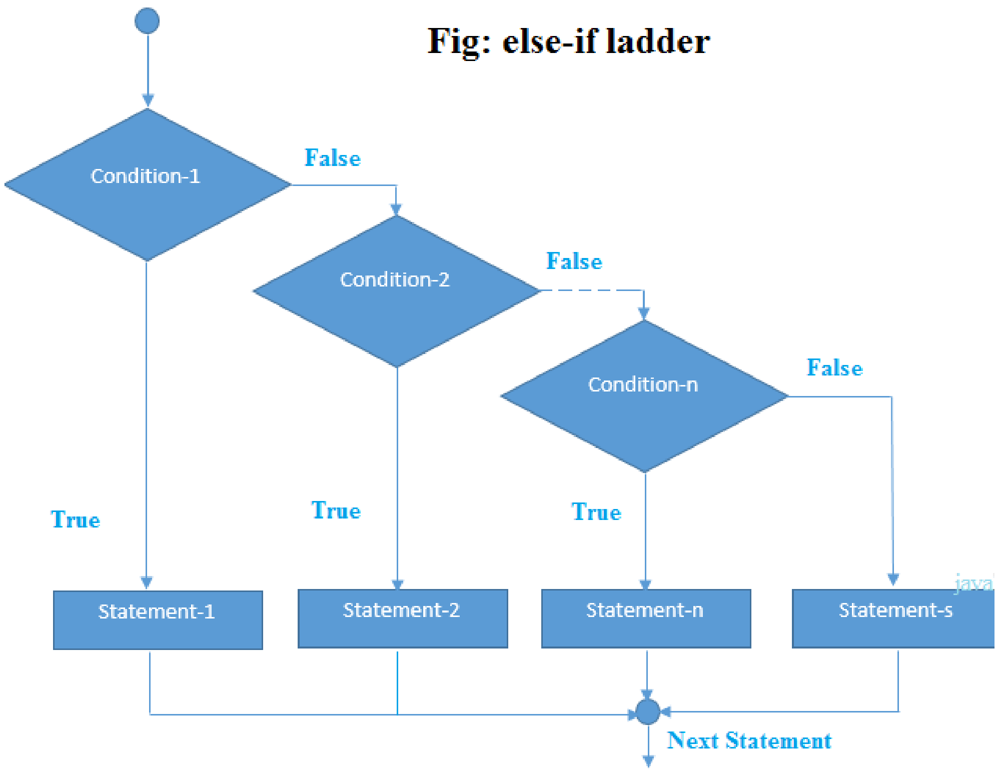


Fig: else-if ladder

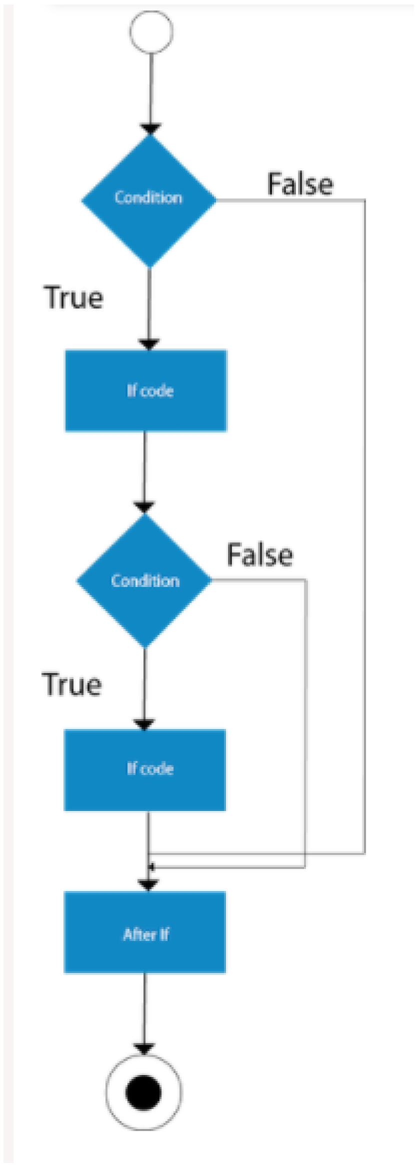
Void main(){

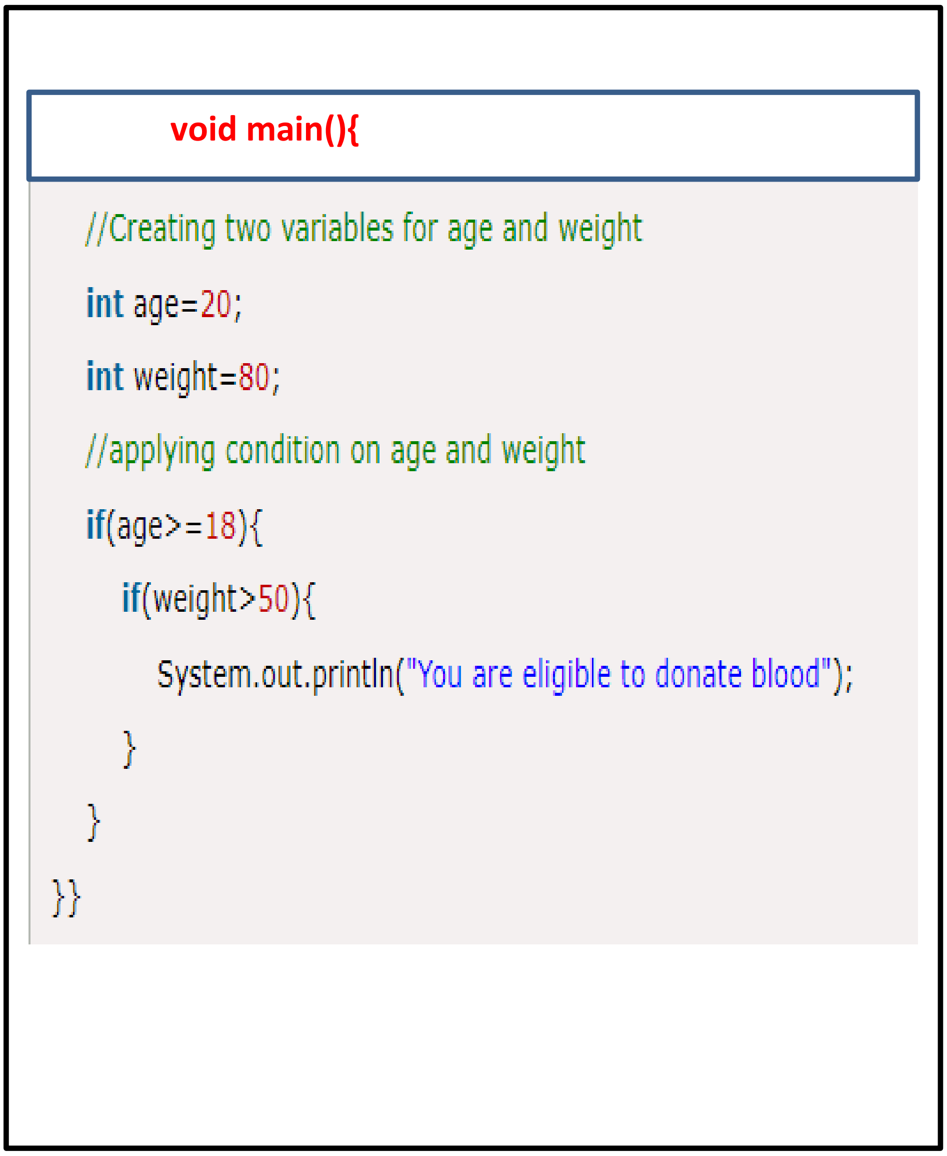
```
    int marks=65;

    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");
    }
}
}
```

4)Nested-if Statement:→

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:➔
```
if(condition){
    //code to be executed
        if(condition){
           //code to be executed
    }
}
```





```
void main(){
  //Creating two variables for age and weight
  int age=20;
  int weight=80;
  //applying condition on age and weight
  if(age>=18){
    if(weight>50){
      System.out.println("You are eligible to donate blood");
    }
  }
}}
```

Drawback: → **As number of conditions increases, the number of if conditions also increases, so complexity also increases. In some situations if the condition specified in the last statement is true then all other statements checks take much time to reach there. The problem is overcomed in switch case statements.**

Switch Statement:→ **It is used when there is many alternatives are given, we have to select one alternative among all.**

SYNTAX:→

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

Rules for switch statement in C language

1) The *switch expression* must be of an integer or character type.

2) The *case value* must be an integer or character constant.

3) The *case value* can be used only inside the switch statement.

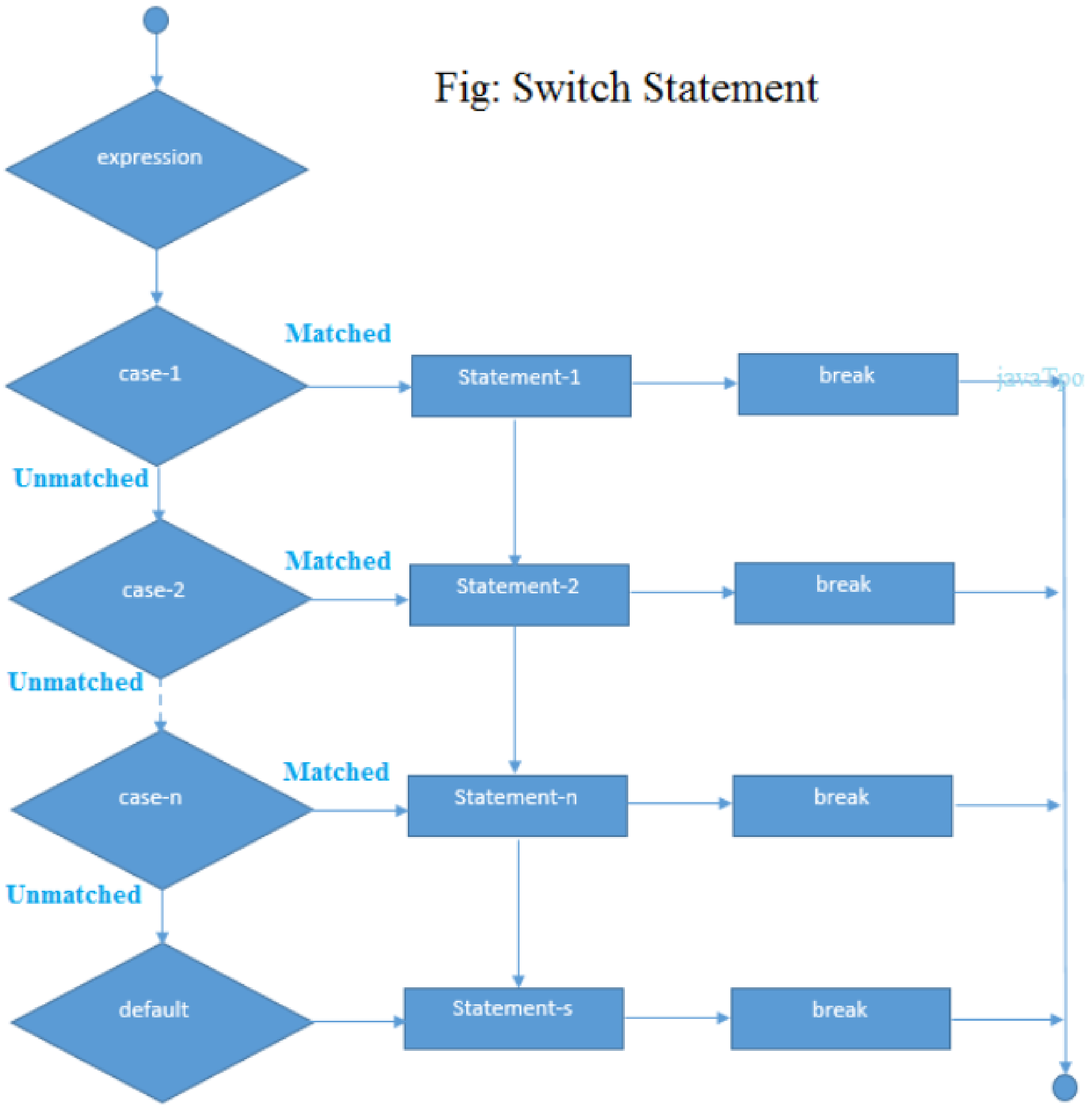
4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Let's try to understand it by the examples. We are assuming that there are following variables.

```
int x,y,z;
char a,b;
float f;
```

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Flowchart of switch statement in C



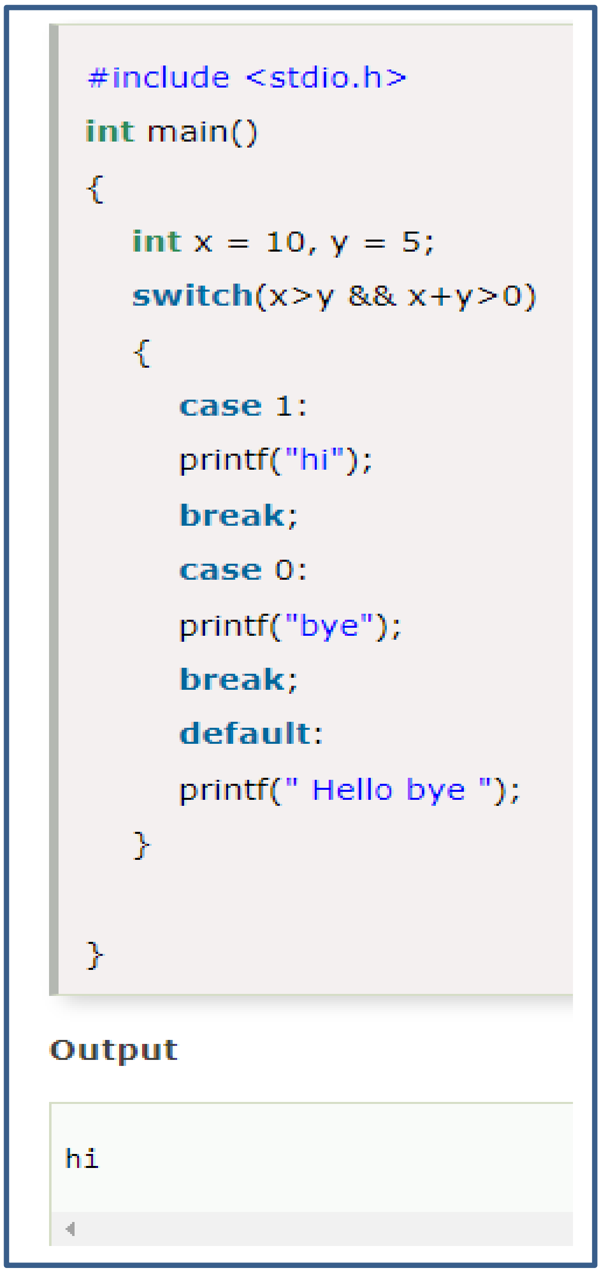
```c
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

OUTPUT:→

```
enter a number:4
number is not equal to 10, 50 or 100
```

OUTPUT:→

```
enter a number:50
number is equal to 50
```



```c
#include <stdio.h>
int main()
{
    int x = 10, y = 5;
    switch(x>y && x+y>0)
    {
        case 1:
        printf("hi");
        break;
        case 0:
        printf("bye");
        break;
        default:
        printf(" Hello bye ");
    }

}
```

Output

```
hi
```

```
#include <stdio.h>
int main()
{
    int x = 10, y = 5;
    switch(x>y && x+y>0)
    {
        case 1:
        printf("hi");
        break;
        case 0:
        printf("bye");
        break;
        default:
        printf(" Hello bye ");
    }

}
```

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

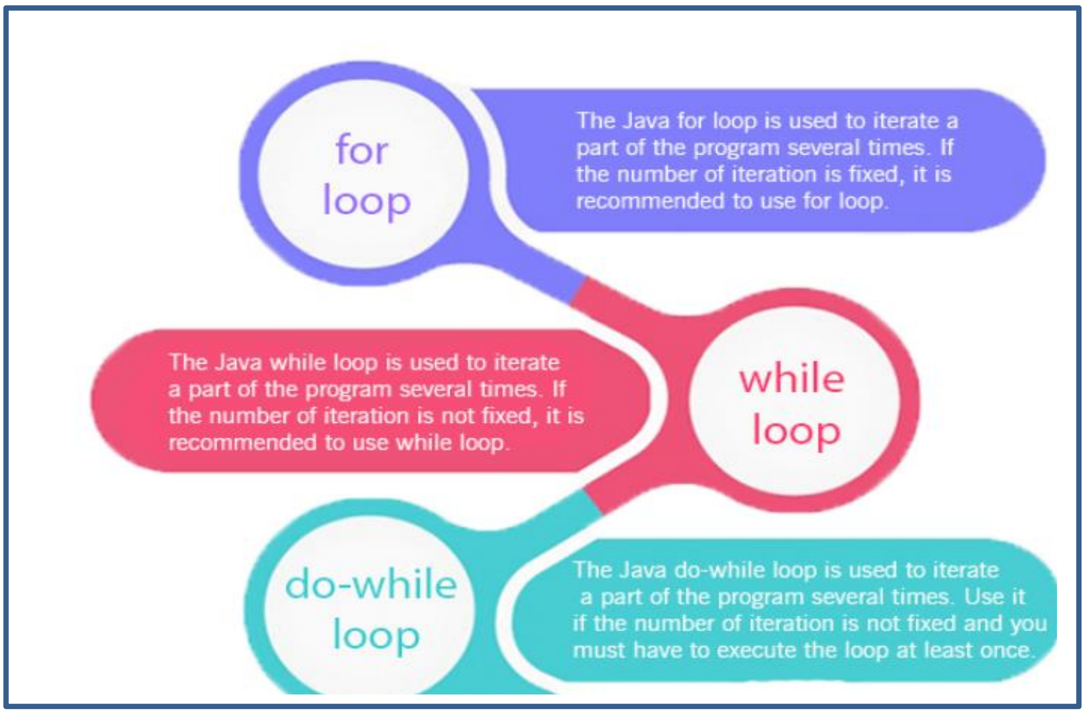
2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while
2. while
3. for


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

For loop in C

The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

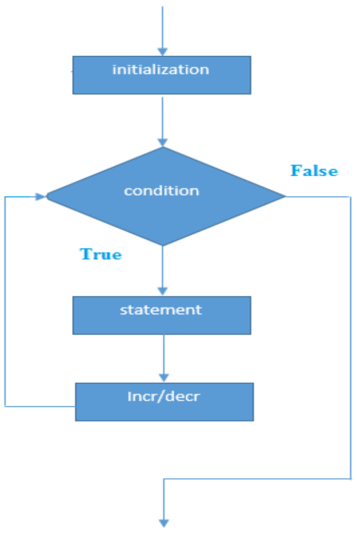
Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Flowchart of for loop in C



C for loop Examples

Let's see the simple program of for loop that prints table of 1.

```
#include<stdio.h>
int main(){
int i=0;
for(i=1;i<=10;i++){
printf("%d \n",i);
}
return 0;
}
```

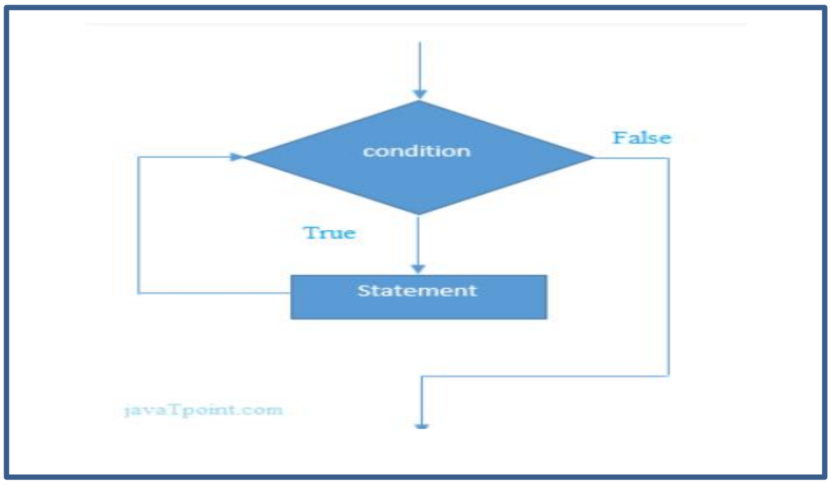
Output

```
1
2
3
4
5
6
7
8
9
10
```

while-loop:→ The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
//code to be executed
}
```



```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d ",i);
i++;
}
return 0;
}
```

Output;-

```
1 2 3 4 5 6 7 8 9 10
```

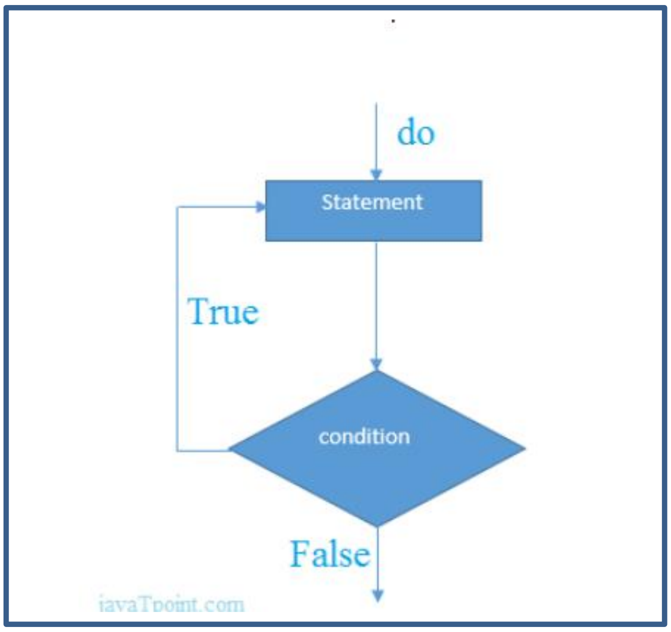
Program to print table for the given number using while loop in C

```
#include<stdio.h>
int main(){
int i=1,number=0,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10){
printf("%d \n",(number*i));
i++;
}
return 0;
}
```

Do-while loop:→ The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

Syntax:→

```
do{
//code to be executed
}while(condition);
```



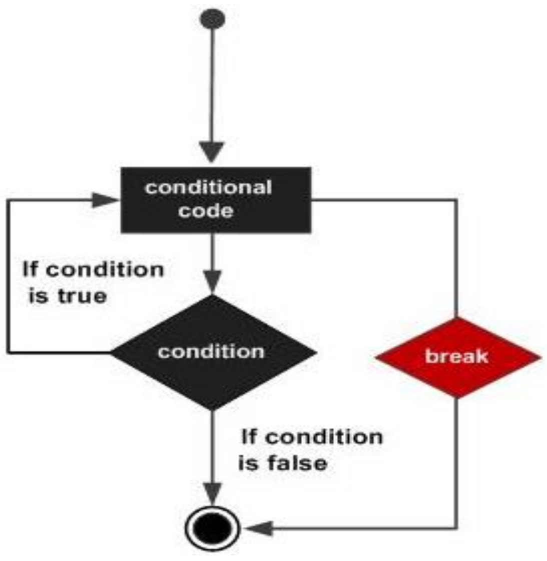
```
#include<stdio.h>
int main(){
int i=1;
do{
printf("%d ",i);
i++;
}while(i<=10);
return 0;
}
```

1 2 3 4 5 6 7 8 9 10

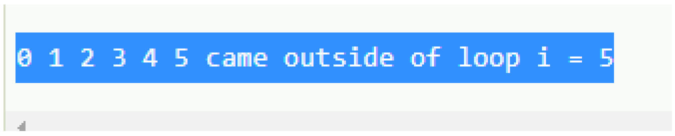
Break statement:→ The **break** statement in Java programming language has the following two usages −

- When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement

Syntax:→break;



```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
   int i;
   for(i = 0; i<10; i++)
   {
      printf("%d ",i);
      if(i == 5)
      break;
   }
   printf("came outside of loop i = %d",i);

}
```



```
0 1 2 3 4 5 came outside of loop i = 5
```

Continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
1. //loop statements
2. continue;
```

```
//some lines of the code which is to be skipped
```

Continue statement example 1

```
#include<stdio.h>
void main ()
{
    int i = 0;
    while(i!=10)
    {
        printf("%d", i);
        continue;
        i++;
    }
}
```

Output

```
infinite loop
```

Program:➔

```
#include<stdio.h>
int main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5){//if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d ",i);
}//end of for loop
return 0;
}
```

OUTPUT:→1 2 3 4 6 7 8 9 10

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complecated.

Syntax:→

```
label:
//some part of the code;
goto label;
```

program:

```c
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;

}
```

```
Enter the number whose table you want to print?10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```