Looping 1

Teaching Team of Fundamentals Programming 2023

Learning Outcome

After finishing this topics, students must be able to:

- Understand the loop algorithm part 1 (for, while, do-while)
- Give a simple example of loopingI
- llustrate the looping case study part 1 using flowchart

Definition

- A loop statement is a command to repeat one or more statements several times
- By using loop statement, we don't have to write one/a set of statements over and over again.
  - In this case, typing errors can be reduced
- Type:
  - Definite loop
  - Indefinite loop

Definite Loop

- A type of loop where **the number of iterations** or **repetitions** is **known** in advance and is definite. This means that you know exactly how many times the loop will execute before it starts running.
- In the case study, usually it is indicated by statement “repeat ___ times”
- Example:
  - Repeat the following statements n times
  - Repeat this statement for each even number between 8 and 26.

Indefinite Loop

- A type of loop where the number of iterations or repetitions is **not predetermined** or **finite**.
- In other words, an indefinite loop will **continue to run indefinitely** until a **specific condition is met** or until it is manually interrupted.
- These loops are typically used when you need a program to keep executing a certain block of code **until a particular condition becomes true**
- The loop will keep running if the condition is TRUE, or until the condition becomes FALSE
- Example:
  - Repeat this statement as long as the number n is not a prime number.
  - Repeat this statement until the user enters a valid integer

Type of Loop Statement

In the Java language, there are three common types of loop statements, which are:

- FOR
- WHILE
- DO-WHILE

FOR

FOR Loop

- FOR is a control flow statement in programming that allows us to **repeatedly execute** a block of code **a specified number** of times
- It is one of the most commonly used loop constructs in many programming languages
- FOR loop typically consists of 3 main part (initialization, condition, iteration)
- FOR structure

```
for (inisialisasi; kondisi; update) statement;
```

or:

```
for (inisialisasi; kondisi; update){
        statement1;
        statement2;
}
```

FOR Loop

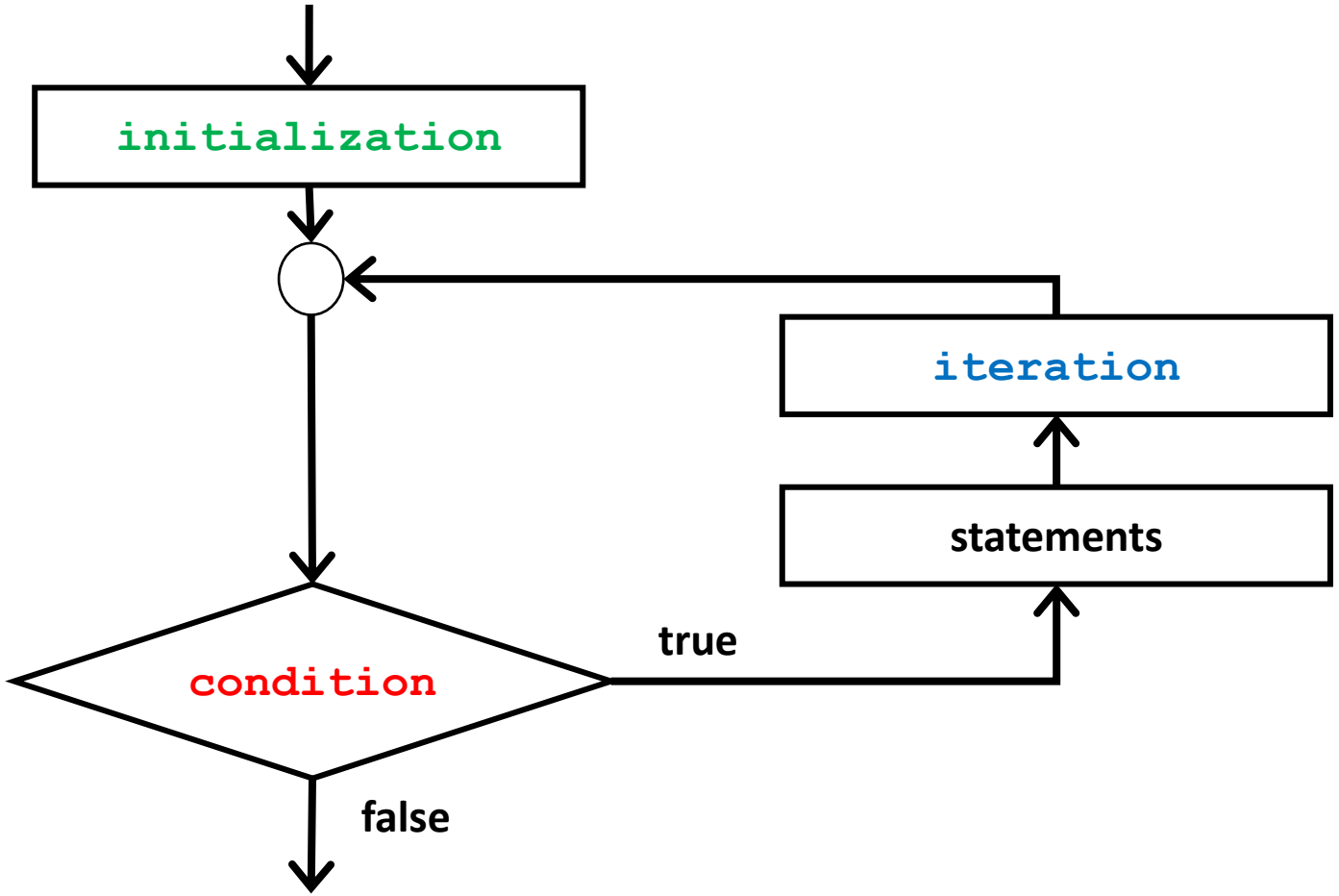
- `initialization`: This part is where you initialize a loop control variable (often referred to as an iterator) to an initial value. It is executed only once at the beginning of the loop.
- `condition`: This part specifies a condition that is checked before each iteration of the loop. If the condition evaluates to true, the loop continues; if it evaluates to false, the loop terminates.
- `iteration`: This part is usually an increment or decrement operation that updates the loop control variable. It is executed at the end of each iteration, just before checking the condition again.

`initialization` and `iteration` are not mandatory

How FOR Loop Works?

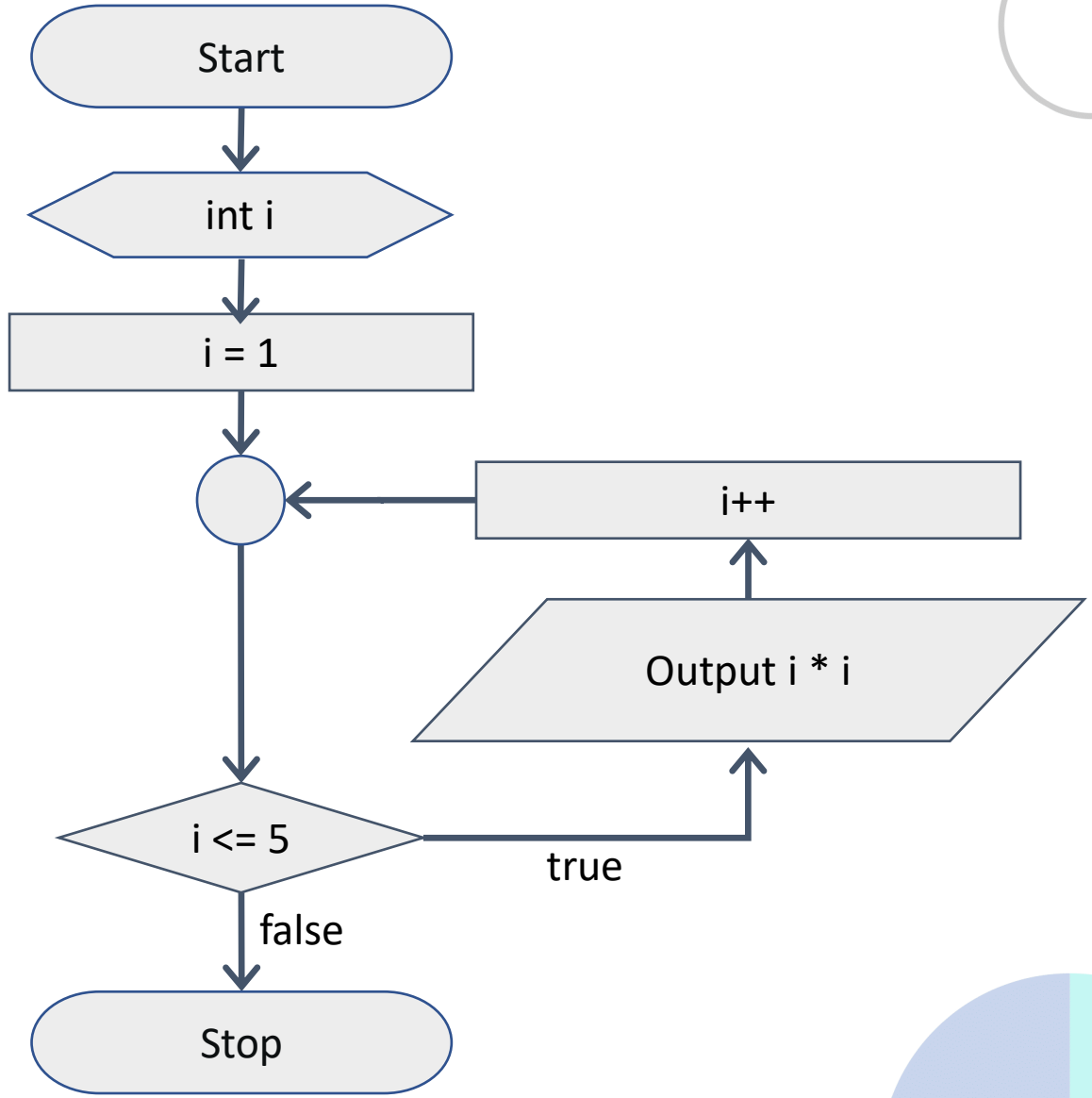
1. The loop is started by executing `initialization`
2. Evaluate `condition`
   - If the `condition` is TRUE, execute all statements in the for block. Once it is finished, execute `iteration`. Then repeat to step 2
   - If it is FALSE, the loop will end (go to after closing bracket } of FOR loop

FOR Flowchart



FOR Example

Create a flowchart and a program code to display numbers and their squares within the range from 1 to 5!



FOR Example

Code Snippet

```
for (int i = 1; i <= 5; i++) {
    System.out.println("Hasil kuadrat dari " + i + " adalah " + (i * i));
}
```

Output

```
Hasil kuadrat dari 1 adalah 1
Hasil kuadrat dari 2 adalah 4
Hasil kuadrat dari 3 adalah 9
Hasil kuadrat dari 4 adalah 16
Hasil kuadrat dari 5 adalah 25
```

Variations of FOR Loop - Variation #1

`initialization` and `iteration` can consist of more than 1 expression, delimited by comma ,

```
for (int i = 1, j = 10; i < j; i++, j--) {
    System.out.printf("%03d -- %03d\n", i, j);
}
```

Output

```
001 -- 010
002 -- 009
003 -- 008
004 -- 007
005 -- 006
```

Variations of FOR Loop - Variation #2

- `initialization` and `iteration` could be blank, based on the needs
- Example:

```
Scanner sc = new Scanner(System.in);
int bil;
boolean berhenti = false;
for (; !berhenti;) {
    System.out.print("Masukkan bilangan: ");
    bil = sc.nextInt();
    System.out.println("Bilangan yang Anda masukkan: " + bil);
    if (bil == 0) {
        berhenti = true;
    }
}
System.out.println("Program berakhir");
```

Output

```
Masukkan bilangan: 4
Bilangan yang Anda masukkan: 4
Masukkan bilangan: 1
Bilangan yang Anda masukkan: 1
Masukkan bilangan: 0
Bilangan yang Anda masukkan: 0
Program berakhir
```

Variations of FOR Loop - Variation #3

Just like the condition in `if`, `condition` in `for` could be a Boolean variable

Contoh:

```
Scanner sc = new Scanner(System.in);
int bil, n;
boolean berhenti = false;
for (n = 0; !berhenti; n++) {
    System.out.print("Masukkan bilangan: ");
    bil = sc.nextInt();
    System.out.println("Bilangan yang Anda masukkan: " + bil);
    if (bil < n) {
        berhenti = true;
    }
}
System.out.println("Program berakhir");
```

Output

```
Masukkan bilangan: 2
Bilangan yang Anda masukkan: 2
Masukkan bilangan: 5
Bilangan yang Anda masukkan: 5
Masukkan bilangan: 3
Bilangan yang Anda masukkan: 3
Masukkan bilangan: 2
Bilangan yang Anda masukkan: 2
Program berakhir
```

WHILE

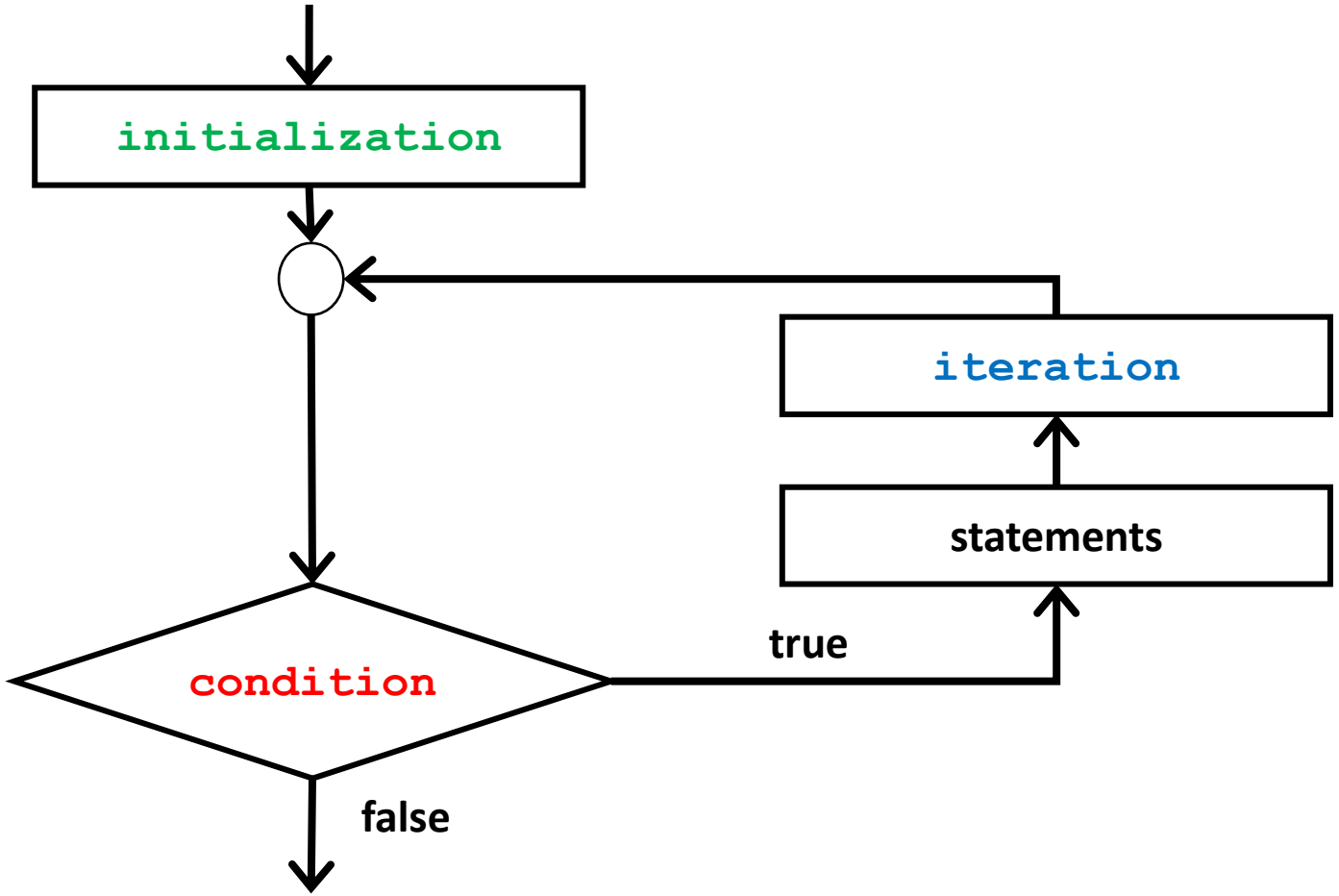
WHILE Loop

- a control flow statement in programming that allows us to repeatedly execute a block of code **as long as** a specified **condition** is **true**.
- Unlike a "for loop" which has a **predetermined number of iterations**, a "while loop" **continues execution indefinitely** until the **condition becomes false**.
- The basic structure of a "while loop" consists of a **condition** that is checked before each iteration. If the condition is true, the loop executes the code block, and after each iteration, it checks the condition again. If the condition ever evaluates to false, the loop terminates, and control moves to the next statement after the loop.

- WHILE Structure

```
inisialisasi;
while (kondisi){
        statement1;
        statement2;
        …
        update;
}
```

WHILE Flowchart



FOR & WHILE Comparison

WHILE

```
initialization;

while (condition)
{
    statement1;
    statement2;
    …
    iteration;
}
```

FOR

```
for (initialization; condition; iteration) {
    statement1;
    statement2;
    …
}
```

Contoh:

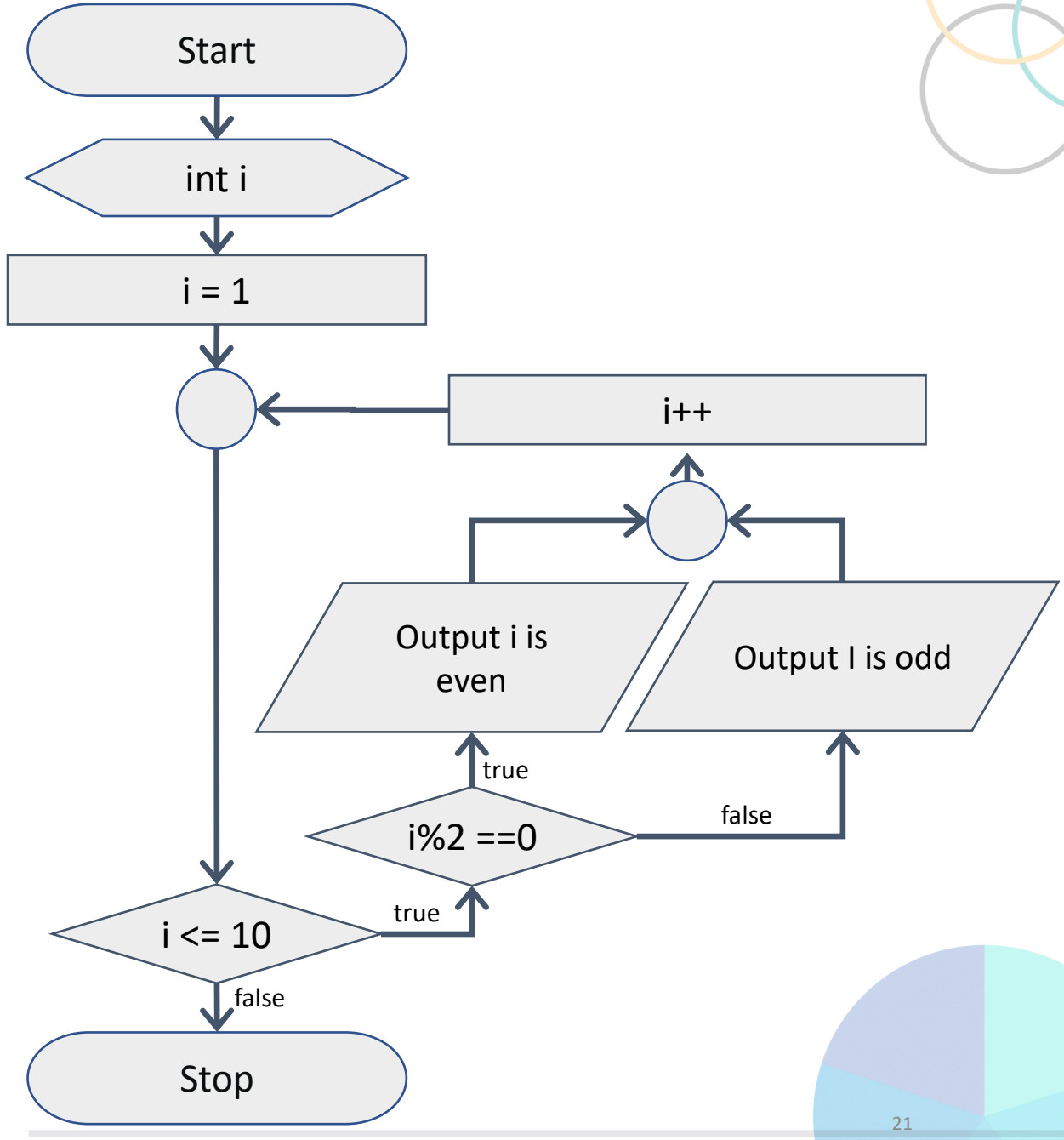
```
int x = 1;

while (x <= 10) {
    ______
    ______
    x++;
}
```

```
for(int x = 1 ;  x <= 10 ;  x++)
    ______
    ______
}
```

WHILE Example

Create a flowchart and program code to display labels for odd and even numbers within the range of values from 1 to 10!



WHILE Example

Code Snippet

```
int i = 1;
while (i <= 10) {
    if (i % 2 == 0) {
        System.out.println("Angka " + i + " adalah bilangan genap");
    } else {
        System.out.println("Angka " + i + " adalah bilangan ganjil");
    }
    i++;
}
```

Output

```
Angka 1 adalah bilangan ganjil
Angka 2 adalah bilangan genap
Angka 3 adalah bilangan ganjil
Angka 4 adalah bilangan genap
Angka 5 adalah bilangan ganjil
Angka 6 adalah bilangan genap
Angka 7 adalah bilangan ganjil
Angka 8 adalah bilangan genap
Angka 9 adalah bilangan ganjil
Angka 10 adalah bilangan genap
```

DO-WHILE

DO-WHILE Loop

- A “DO-WHILE" loop is a control flow statement in programming that is **similar** to a “WHILE loop,"
- There is one key difference: in a " DO-WHILE " loop, the **condition is checked after** the loop body has been **executed at least once**. This means that the code inside the loop block is guaranteed to run at least once, regardless of whether the condition is initially true or false.
- DO-WHILE: the condition will be evaluated after the loop block
- WHILE: the condition is evaluated before the loop block

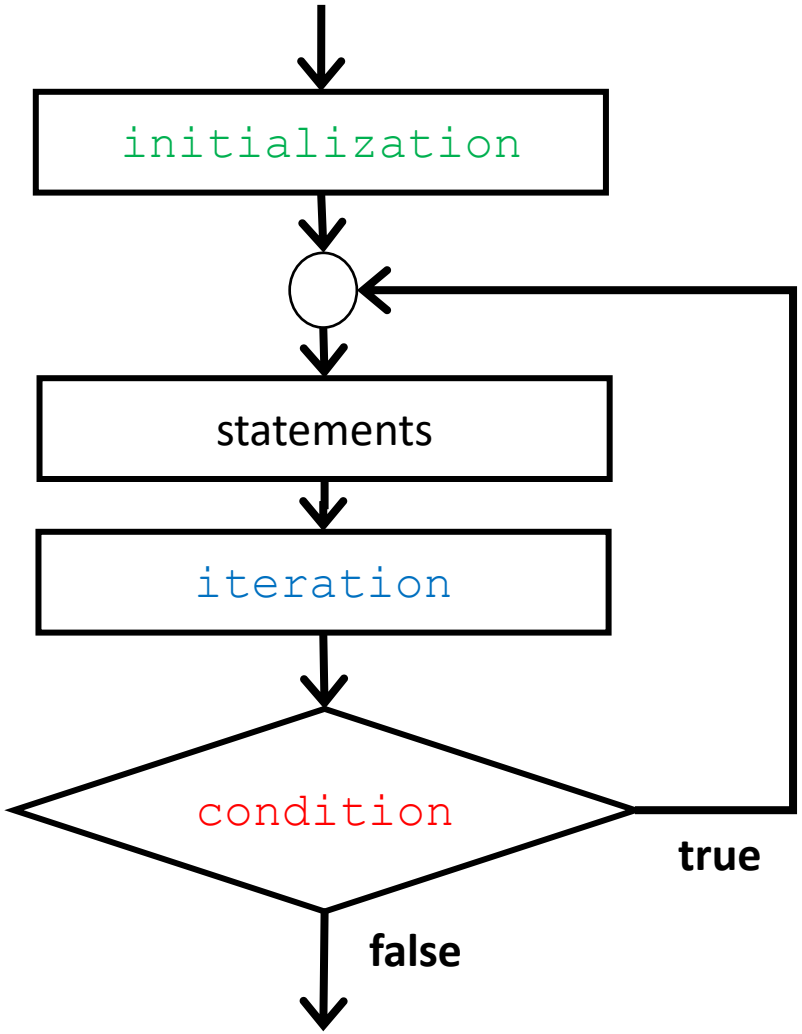
DO-WHILE Loop

- DO-WHILE Structure

```
Initialization;
do{
        statement1;
        statement2;
        …
        iteration;
} while (condition);
```

Execute the statement at least once.
As long as the `condition` is true, the loop will continue to run

DO-WHILE Flowchart



DO-WHILE Example

Code Snippet

```
int x = 0;
do {
    System.out.println(x);
} while (++x <= 8);
System.out.println("Program berhenti");
```

Output

```
0
1
2
3
4
5
6
7
8
Program berhenti
```

Code Snippet

```
int x = 10;
do {
    System.out.println(x);
} while (++x <= 8);
System.out.println("Program berhenti");
```

Output

```
10
Program berhenti
```

Infinite Loop

Infinite Loop

- An "infinite loop" is a type of loop in computer programming that **continues** to **execute endlessly** without a predefined exit condition.
- In other words, it's a loop that **runs forever**, and there's no natural termination point.
- Infinite loops are **typically considered errors** in programming because they can cause a program to **hang** or **become unresponsive**
- Thus, when executing statements within a loop, there must be a **condition** that makes the condition evaluate to **FALSE**
- Otherwise, the **condition** will be true forever, and the loop will run forever as well (**infinite loop)**

Infinite Loop Example

Code Snippet

```
int hitung = 1;
while (hitung <= 25) {
    System.out.println(hitung);
    hitung = hitung - 1;
}
```

Output

```
1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 -16 -17 -18 -19 -20 -21 -22
-23 -24 -25 -26 -27 -28 -29 -30 -31 -32 -33 -34 -35 -36 -37 -38 -39 -40 -41 -42 -43
```

The loop will run continuously until the user stops it

Ways to Stop A Loop

How to Stop A Loop

Some ways to stop loop in interactive programs include:

- Adding a Sentinel or delimiter with a special code
- Adding a question that will decide whether the loop continues or not

Adding Sentinel

- A sentinel is a value that indicates **the end of user input**
- **A sentinel loop** represents a loop that will continue running until the sentinel value is encountered.

Adding Sentinel – Example

Write a program to get user input (positive integers) until the user enters -1 to stop. Print the sum and average of the entered numbers

```
Scanner sc = new Scanner(System.in);
int jumlah = 0, counter = 0, angka;
float rata = 0;
do {
    System.out.print("Masukkan integer positif (-1 untuk berhenti): ");
    angka = sc.nextInt();
    if (angka >= 0) {
        jumlah += angka;
        ++counter;
    }
} while (angka != -1);
rata = jumlah / counter;
System.out.printf("Jumlah dari %d angka adalah %d\n", counter, jumlah);
System.out.printf("Rata-rata dari %d angka adalah %.3f\n", counter, rata);
```

Output

```
Masukkan integer positif (-1 untuk berhenti): 10
Masukkan integer positif (-1 untuk berhenti): 20
Masukkan integer positif (-1 untuk berhenti): 30
Masukkan integer positif (-1 untuk berhenti): 40
Masukkan integer positif (-1 untuk berhenti): 50
Masukkan integer positif (-1 untuk berhenti): -1
Jumlah dari 5 angka adalah 150
Rata-rata dari 5 angka adalah 30.000
```

Adding Question

- **Questions** are used to provide users with the option to continue the loop
- If the condition in the loop evaluates to TRUE based on the user's answer to the question, then the loop continues
- Example:
  - Will you continue the loop?
  - Will you add a new item?

Adding Question – Example

Write a program code to receive input for a list of customer names. Print the total number of customers entered

```
Scanner sc = new Scanner(System.in);
String nama;
char jawab;
int jml = 0;
do {
    System.out.print("Masukkan nama pelanggan: ");
    nama = sc.nextLine();
    jml++;
    System.out.print("Apakah Anda ingin memasukkan nama pelanggan baru (Y/T)? ");
    jawab = sc.nextLine().charAt(0);
} while (jawab == 'y' || jawab == 'Y');
System.out.println("Jumlah pelanggan yang Anda masukkan = " + jml);
```

Output

```
Masukkan nama pelanggan: Afi
Apakah Anda ingin memasukkan nama pelanggan baru (Y/T)? y
Masukkan nama pelanggan: Brian
Apakah Anda ingin memasukkan nama pelanggan baru (Y/T)? Y
Masukkan nama pelanggan: Dewi
Apakah Anda ingin memasukkan nama pelanggan baru (Y/T)? t
Jumlah pelanggan yang Anda masukkan = 3
```

Statement
BREAK & CONTINUE

BREAK

- A "**break** statement" is a control flow statement used in programming languages to **exit** or **terminate** a **loop prematurely**, before it reaches its natural exit point (i.e., before the loop condition becomes false).
- When a **break** statement is encountered within a loop, it immediately terminates the loop and transfers control to the statement immediately after the loop.
- **Break** statements are often used to add conditional exit points within loops. They are particularly useful when you want to exit a loop based on certain conditions that may not be known when the loop is initially defined.
- Break statement is also used in switch-case block to exit from switch block

BREAK Example

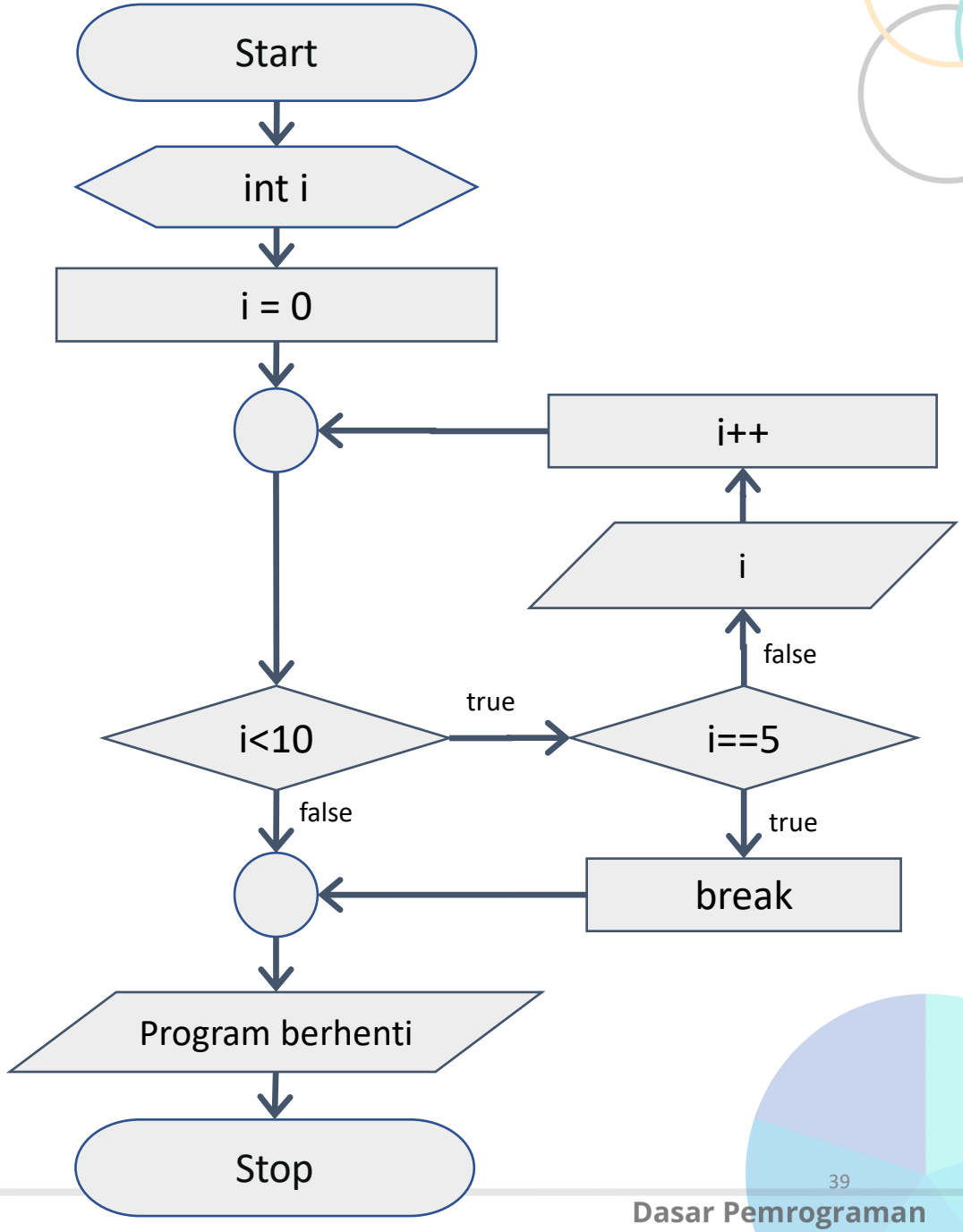
Code Snippet

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.print(i + " ");
}
System.out.println("\nProgram berhenti");
```

Exit from loop

Output

```
0 1 2 3 4
Program berhenti
```



CONTINUE

- A "continue statement" is a control flow statement used in programming to **skip the current iteration** of a loop and **proceed to the next iteration**.
- When a **continue** statement is encountered within a loop, it causes the program to **skip** the remaining code in the current iteration and jump back to the beginning of the loop for the **next iteration**.
- The **continue** statement is typically used in loops, such as for loops and while loops, to control the flow of execution based on certain conditions without exiting the loop entirely.
- It allows you to bypass specific iterations while continuing the overall loop.

CONTINUE

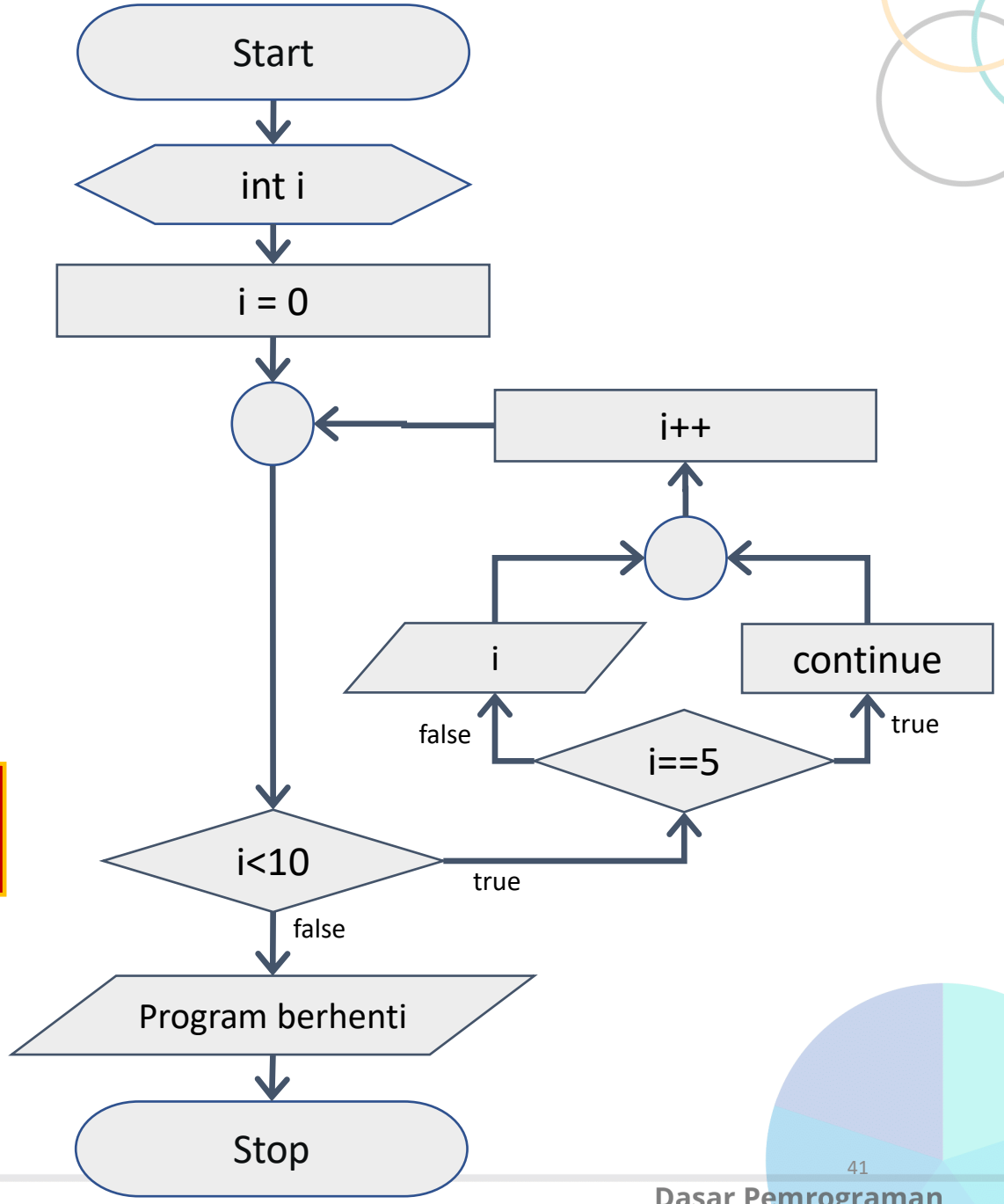
Code Snippet

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue;
    }
    System.out.print(i + " ");
}
System.out.println("\nProgram berhenti");
```

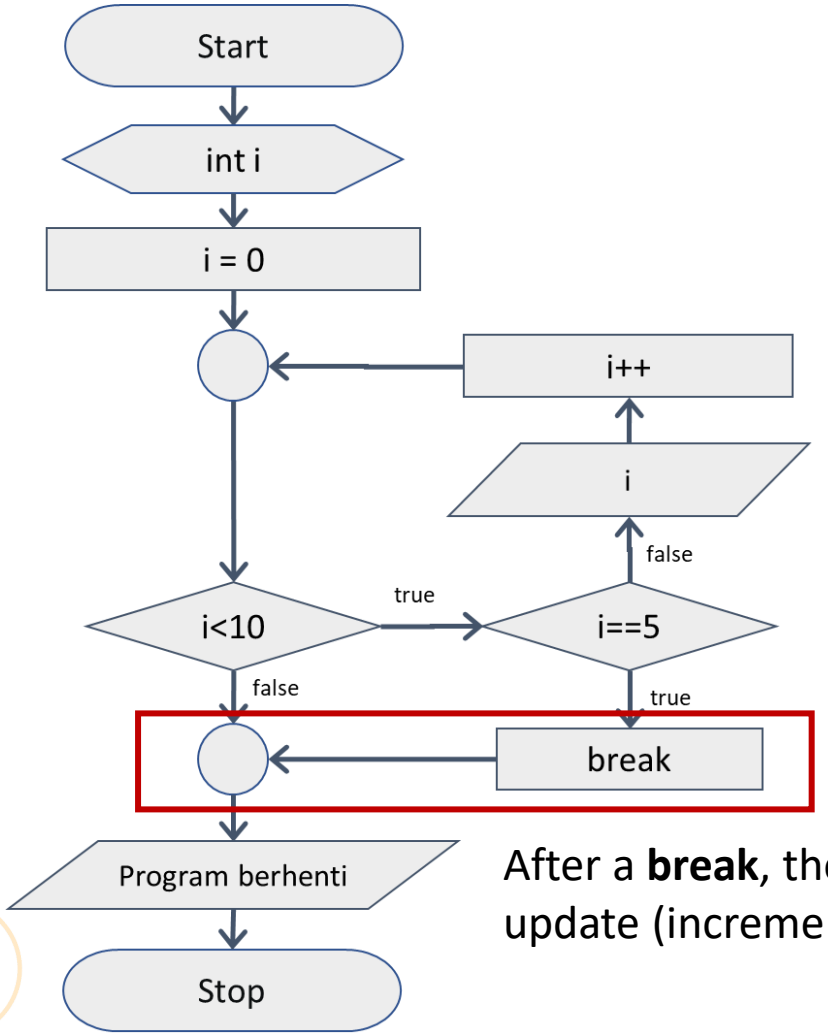
Skip the remaining code and jump to the next iteration

Output

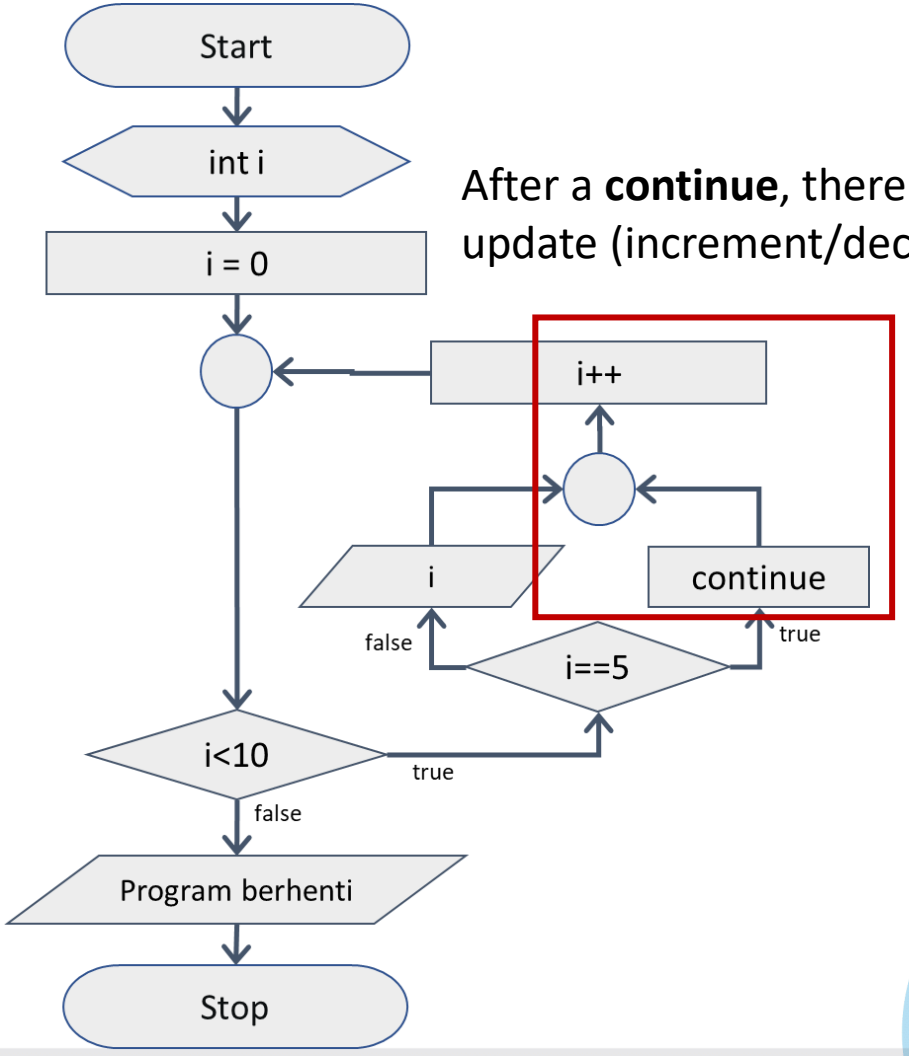
```
0 1 2 3 4 6 7 8 9
Program berhenti
```



BREAK & CONTINUE difference in a Flowchart



After a **break**, there is no value update (increment/decrement)



After a **continue**, there is a value update (increment/decrement)

Exercise

Create a flowchart for the following statements using FOR, WHILE, or DO-WHILE:

a. Users input the names and genders of 30 students in a class. Display only the names of female students.
b. Display the sum of numbers from 25 to 1
c. Display the series of numbers from 1 to 50, excluding multiples of 3 (1 2 4 5 7 8 10 ... 47 49 50).

Assignment

1. Identify in accordance with your respective project which features require the concept of loops
2. Determine the appropriate form of the loop (FOR, DO-WHILE, WHILE-DO).
3. Use a sentinel/BREAK/CONTINUE if necessary.
4. Create an algorithm in the form of a flowchart.