CSE1121
Structured & OOP Language

Decision Making and Looping

Sumaya Kazary
Assistant Professor
Department of Computer Science and Engineering
Dhaka University of Engineering & Technology, Gazipur

Acknowledgement
Thanks to the authors of all the books and online tutorials used in this slide.

Decision Making and Looping

- A loop lets you write a very simple statement to produce a significantly greater result simply by repetition.
  - for statement
  - while statement
  - do while statement
- Two new statements used with loops
  - break and continue

Repetition in Programs

- In most software, the statements in the program may need to repeat for many times.
  - e.g., calculate the value of *n*!.
  - If *n* = 100, it's not elegant to write the code as 1*2*3*…*100.
- ***Loop*** *is a control structure that repeats a group of steps in a program.*
  - ***Loop body*** *stands for the* ***repeated statements****.*

The *for* Repetition Structure

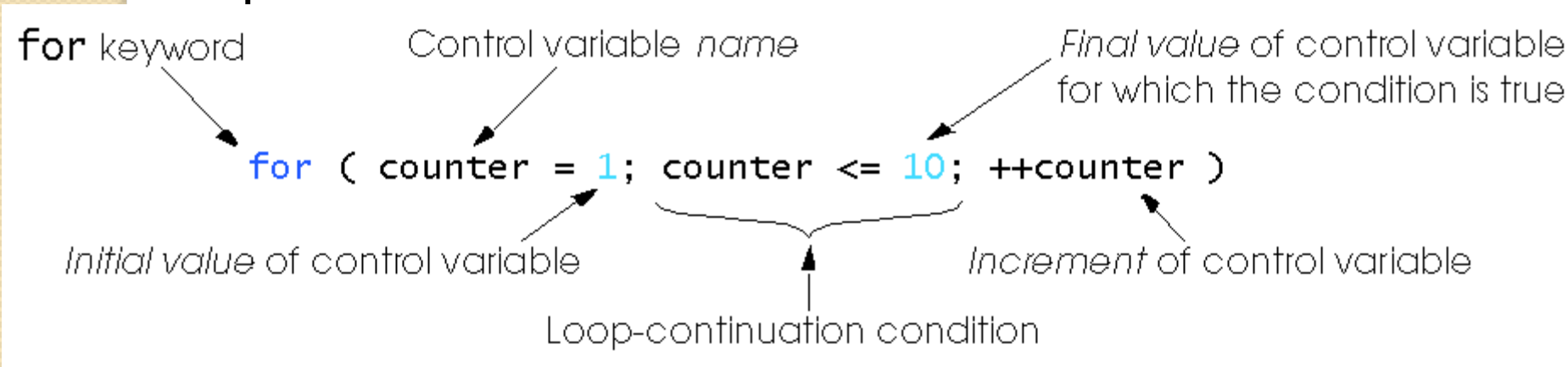
- The *for* loop is (for many people, anyway ☺) the easiest C loops to understand.
  - All its loop-control elements are gathered in one place,
- In the other loop constructions, they are scattered about the program
  - which can make it harder to unravel how these loops work.

The for Repetition Structure

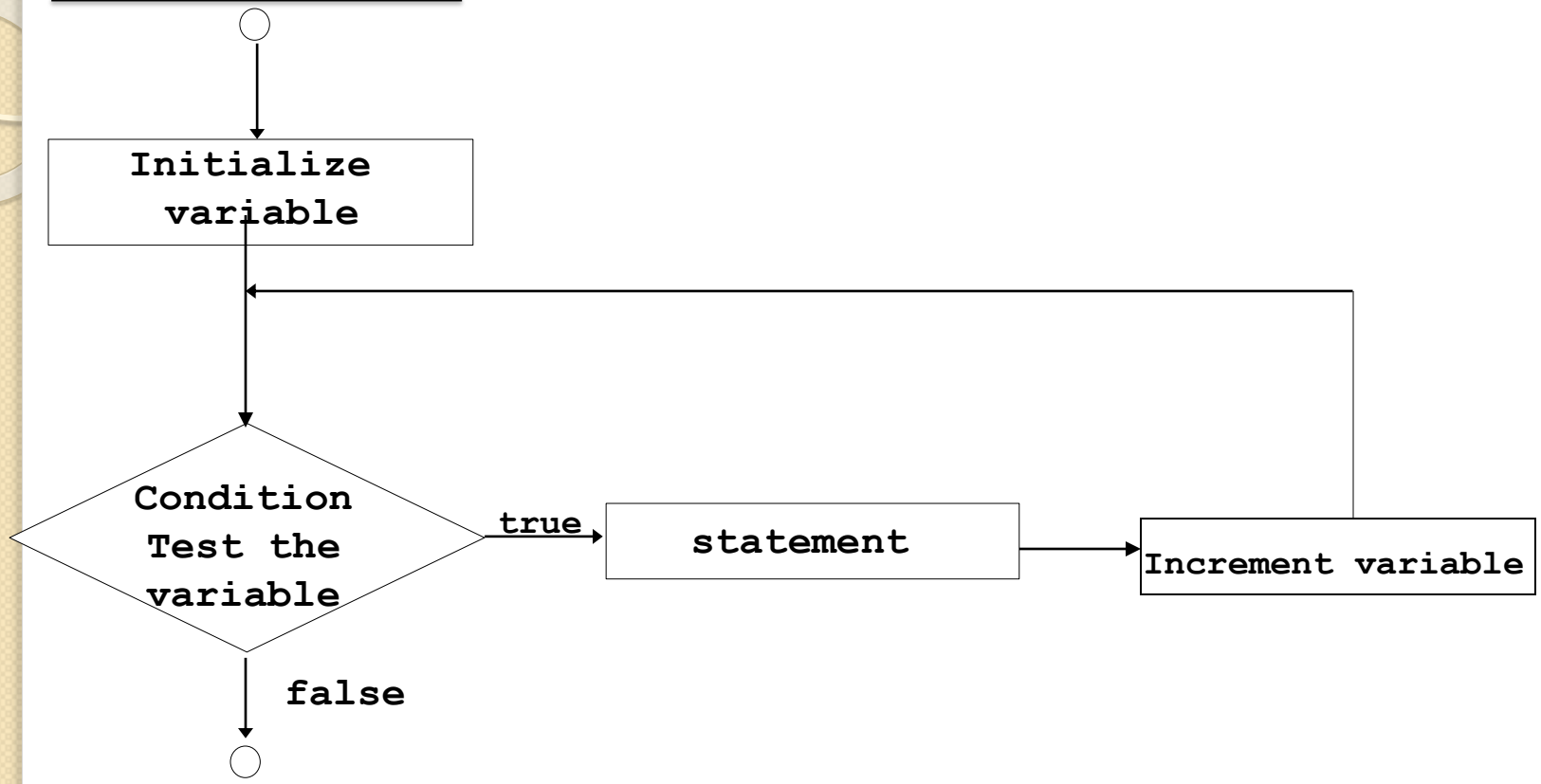
- The general format when using **`for`** loops is

```
for(initialization; LoopCondTest; increment)
    statement(s) ;
```

- Example:



Flowchart : for Repetition Structure



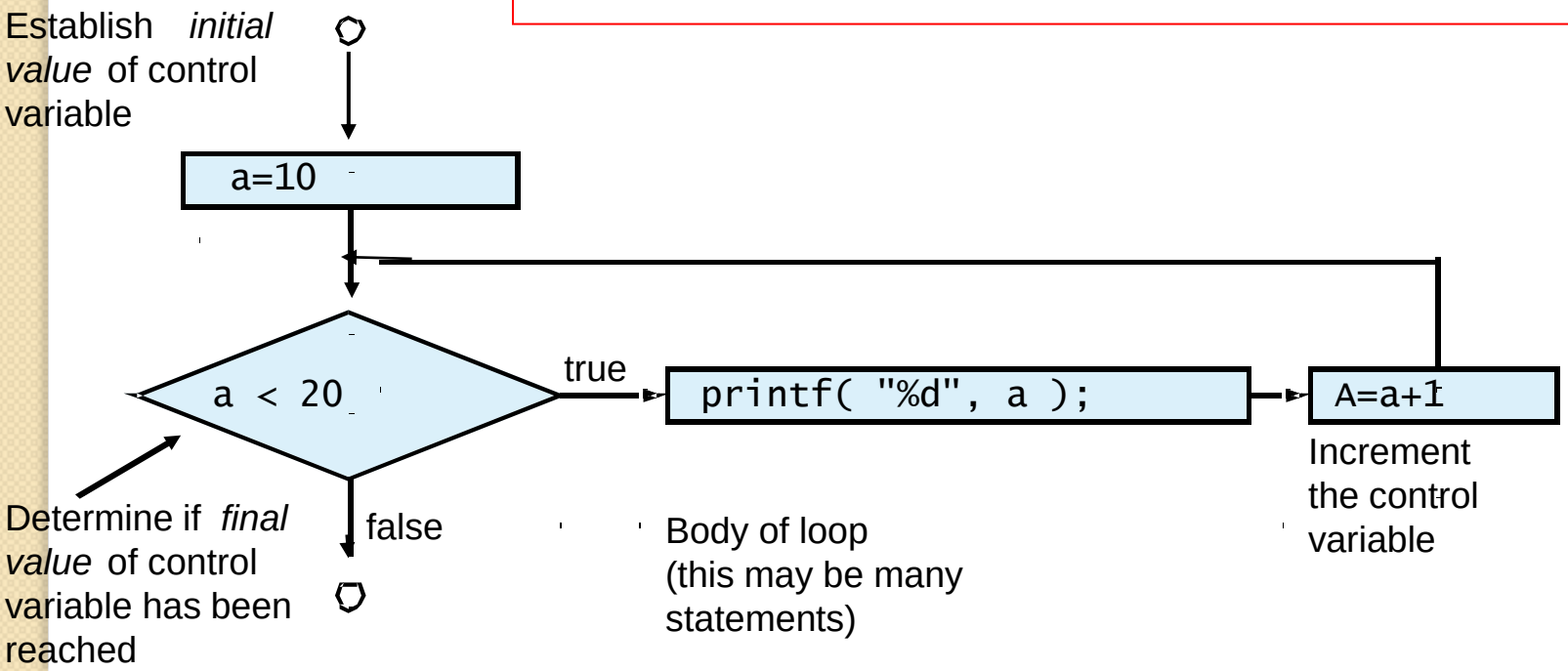
The for Repetition Structure

- The variable initialization allows you to either declare a variable and give it a value or give a value to an already existing variable.
- The condition tells the program that while the conditional expression is true the loop should continue to repeat itself.
- The variable update section is the easiest way for a *for loop* to handle changing of the variable.
  - It is possible to do things like x++, x = x + 10 etc.
- *Notice that a semicolon separates each of these sections, that is important.*

Example

```
int main ()
{     int a;
      for( a = 10; a < 20; a = a + 1 )
      {   printf("value of a: %d\n", a);
       }
return 0;
}
```

Here, a is set to 10, while counter is less than 20 ,it calls printf( ) to display the value of the variable a, and it adds 1 to a until the condition is met.



Example : Print 1 to 100

```
int main(){
  int count;
  for(count=1; count<= 100; count++)
      printf("%d\t ", count);
  printf("\n");
  return 0;
}
```

Exercise

- *Determine the number of times that each of the following for loops are executed.*

```
for (k=3; k<=10; k++) {
    statements;
}

for (k=3; k<=10; ++k)
{
    statements;
}

for(count=2;count<=5; count++) {
    statements;
}
```

$$\left\lfloor \frac{final - initial}{increment} \right\rfloor + 1$$

Some Variations of *for* Loop

- ✓ `for (x=0;((x>3) && (x<9)); x++)`
- ✓ `for (x=0,y=4;((x<3) && (y<9)); x++,y+=2)`
- ✓ `for (x=0,y=4,z=4000; Z;  z/=10)`

Some Interesting points:

- An interesting trail of the *for* loop is that-

  Pieces of the loop definition need not be there.

e.g; `for(x=0;x!=123;)  scanf("%d", &x);`

[Here, the update is blank!!!!!!!]

Each time the loop repeats, x is tested to see if it equals 123;

If x=123, then the loop condition is false and the loop ends.

- To create an infinite loop:-

```
for(; ;) printf ("You are locked!!!\n");
```

If the condition is absent, it's assumed to be TRUE!!!!!!!!

- *for* loop with no bodies:- [to create time delay]

```
for(i=0;  i<100;  i++);
```

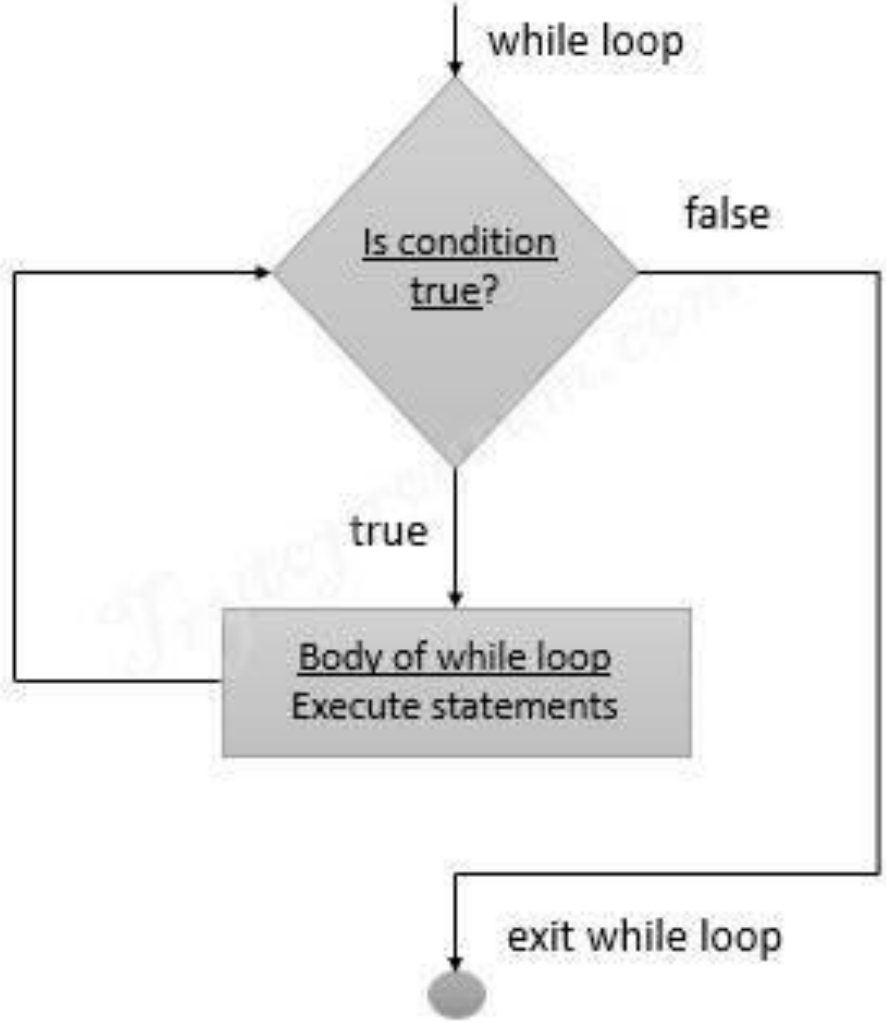
The `while` Statement in C

- The syntax of **`while`** statement in C:

```
while (loop repetition condition)
    statement;
```

  - **`Loop repetition condition`** is the condition which controls the loop.
  - The ***`statement`*** is repeated as long as the loop repetition condition is **true**.
- A loop is called an **infinite loop** if the loop repetition condition is always true.

The while Control Structure



The while Repetition Structure

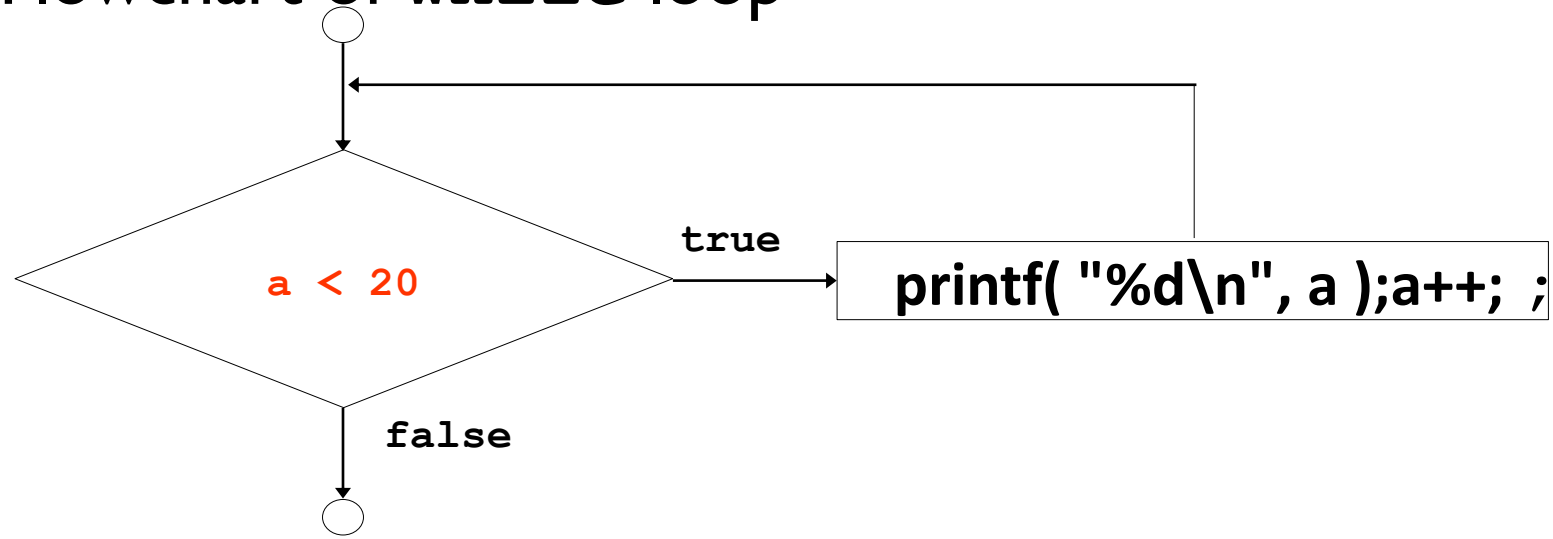
- Notice that a while loop is like a stripped-down version of a *for* loop-- it has no initialization or update section.

```
initialization;
while ( loopCondition Test){
        statement;
        increment;
}
```

- An empty condition is not legal for a while loop as it is with a for loop.

The while Repetition Structure

- Flowchart of **`while`** loop




```
int main ()
{    int a = 10;
     while( a < 20 )
     {
        printf("value of a: %d\n", a);
        a++;
     }
return 0; }
```

The do/while Repetition Structure

- The **do/while** repetition structure is almost similar to the **while** structure, Syntax:-

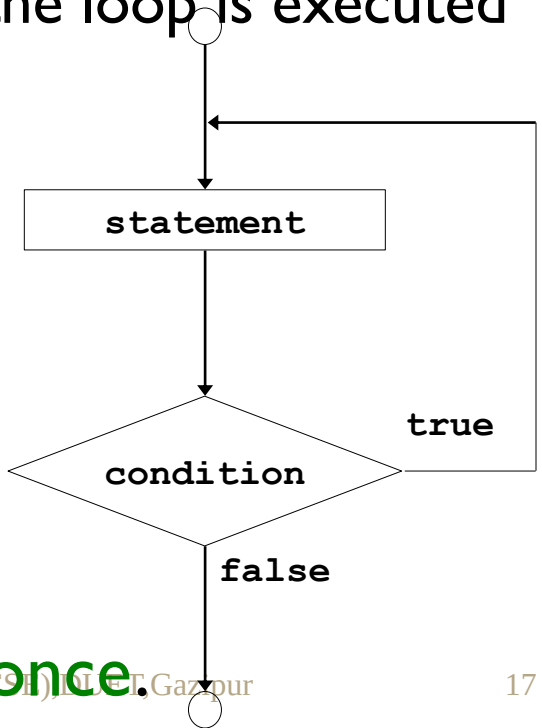
```
do {
    statement;
} while ( condition );
```

  - Condition is tested after the body of the loop is executed

- Example (a = 10):

```
do
{ printf("value of a: %d\n", a);
  a = a + 1;
 }while( a < 20 );
```

*This prints the integers from **10** to **19***



- All actions are performed at least once.

while Vs. do..while

- A do..while loop is almost the same as a while loop except that the loop body is guaranteed to execute at least once.

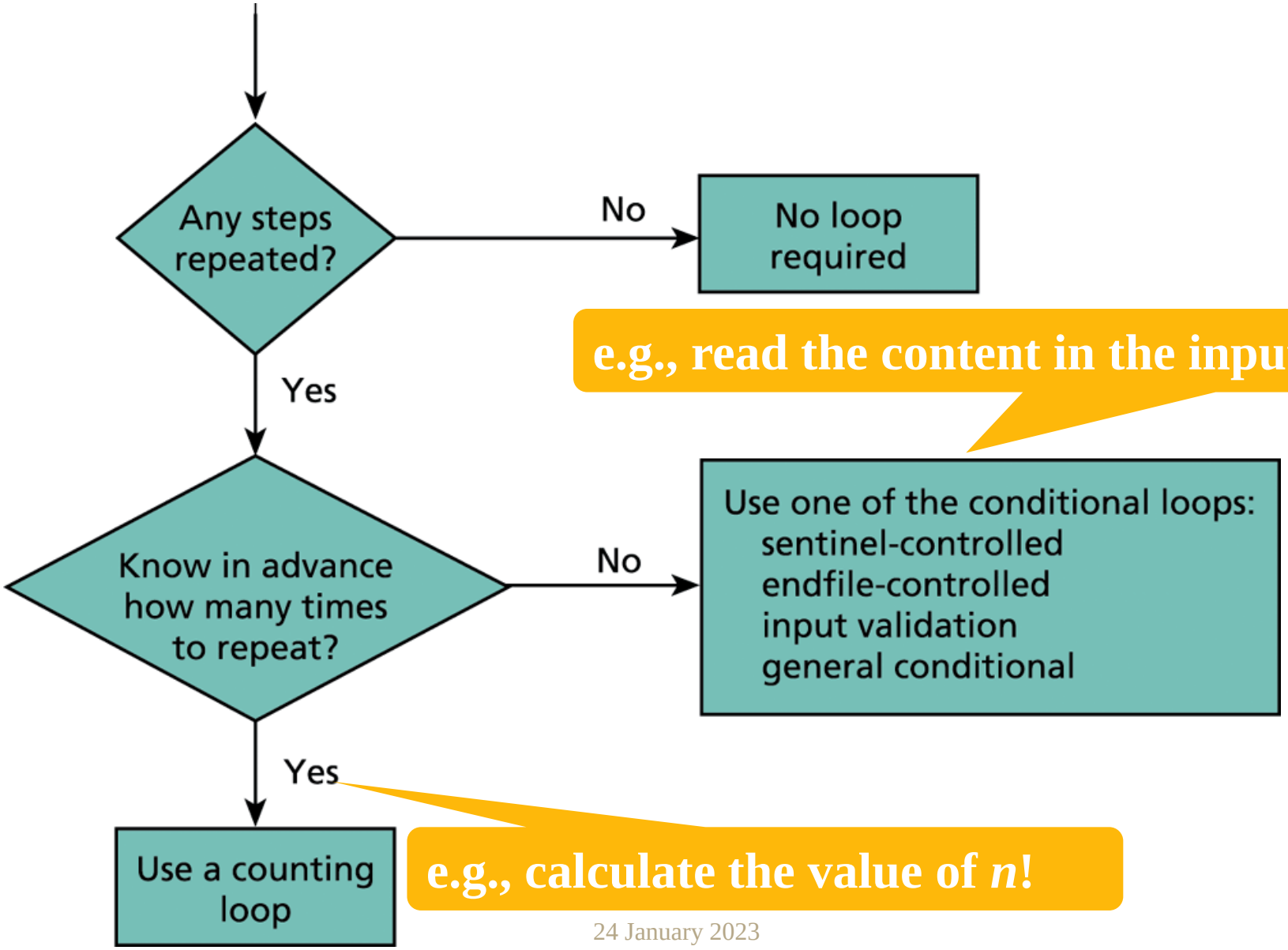
- A **while loop says** "Loop while the condition is true, and execute this block of code",

  A ***do..while* loop says** "Execute this block of code, and then continue to loop while the condition is true".

- A common error is to forget that a do..while loop must be terminated with a semicolon.

- Notice that do..while will execute once, because it automatically executes before checking the condition.

Flow Diagram of Loop Choice Process



e.g., read the content in the input

e.g., calculate the value of *n*!

Counter-Controlled Repetition

- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value.

- Definite repetition
  - Number of repetitions is known

- Example

  A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
  - A loop control variable (or loop counter).
  - The initial value of the control variable.
  - The condition that tests for the final value of the control variable (i.e., whether looping should continue).
  - The update [increment (or decrement)] by which the control variable is modified each time through the loop.

- Example:

```
int counter =1;     /*initialization*/
while (counter <= 10){ /* condition*/
   printf("%d\n", counter);
   ++counter;        //increment
  }
```

Sentinel-Controlled Loops

- Sometimes we may not know how many times the loop will repeat.
- One way to do this is to choose a **sentinel value** as an end marker.
  - The loop exits when the **sentinel value** is read.
- A sentinel controlled loop is also called an indefinite repetition loop because the number of iterations is not known before the loop starts executing.

Example

```
int main(){
int number, sum=0;
do{
     printf("\n Input a number:");
     scanf( "%d",  &number);
     sum+=number;
  }while(number >= 0);
return 0;
}
```

The **break** and **continue** Statements

- **Break**
  - Causes immediate exit from a **`while`**, **`for`**, **`do/while`** or **`switch`** structure
  - Program execution continues with the first statement after the structure
  - Common uses of the **`break`** statement:
    - Escape early from a loop
    - Skip the remainder of a **`switch`** structure

break statement

- break;
  - terminates loop
  - execution continues with the first statement following the loop

```
int main ()
 {  int a = 10;
    while( a < 20 )
    {    printf("value of a: %d\n", a);
         a++;
         if( a > 15)  break;
    }
return 0;
}
```

Continue Statement

- Skips the remaining statements in the body of a **`while`**, **`for`** or **`do/while`** structure and proceeds with the next iteration of the loop
- In **`while`** and **`do/while`**, the loop-continuation test is evaluated immediately after the **`continue`** statement is executed
- In the **`for`** structure, the increment expression is executed, then the loop-continuation test is evaluated
- Essentially, the continue statement is saying
  - "This iteration of the loop is done, let's continue with the loop without executing whatever code comes after me."

The continue Statement

```
int main ()
 {  int a = 10;
    while( a < 20 )
    {    printf("value of a: %d\n", a);
         a++;
         if( a == 15)
         {  a++;
            continue;
          }
    }
return 0;
}
```

The numbers 10 through 19 are printed except for 15.

Thank You