Loops / Repetition Statements

- ***Repetition statements*** **allow us to execute a statement multiple times**
- **Often they are referred to as *loops***
- **C has three kinds of repetition statements:**
  - **the *while loop***
  - **the *for loop***
  - **the *do loop***
- **The programmer should choose the right kind of loop for the situation**

There are three loop constructs in C

- **while loop**
- **for loop**
- **do-while loop (or do loop for short)**
- **Loops = repetition statements**

Example 1: Fixing Bad Keyboard Input

- **Write a program that refuses to accept a negative number as an input.**
- **The program must keep asking the user to enter a value until he/she enters a positive number.**
- **How can we do this?**

Example 2: Grade of several students

- **Write a program that continuously calculates the grade of all students marks and stop when the user wants.**

- **After calculating one student's grade (from his marks the program must keep asking the user whether he likes to continue or not.**

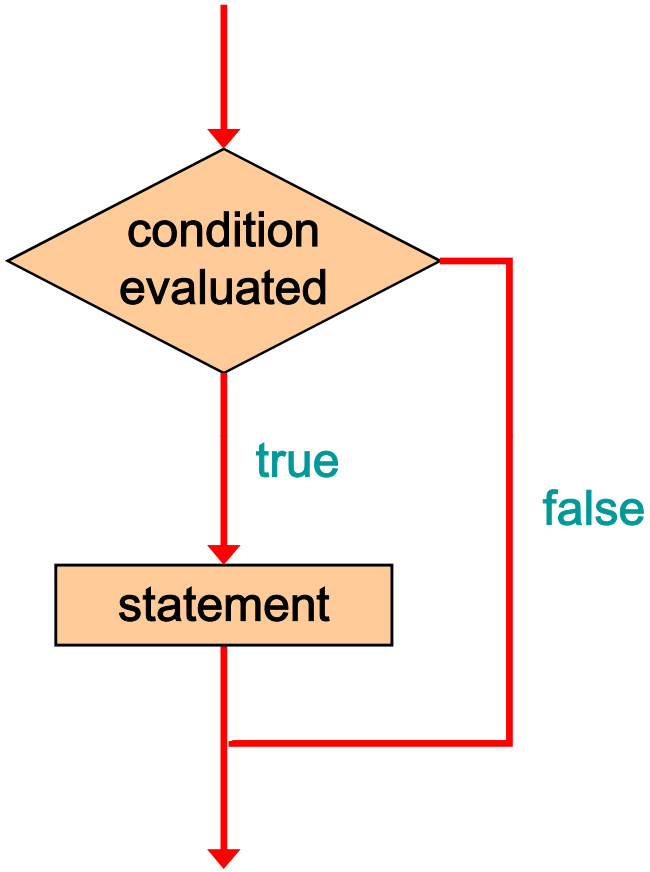
- **How can we do this?**

while Loop

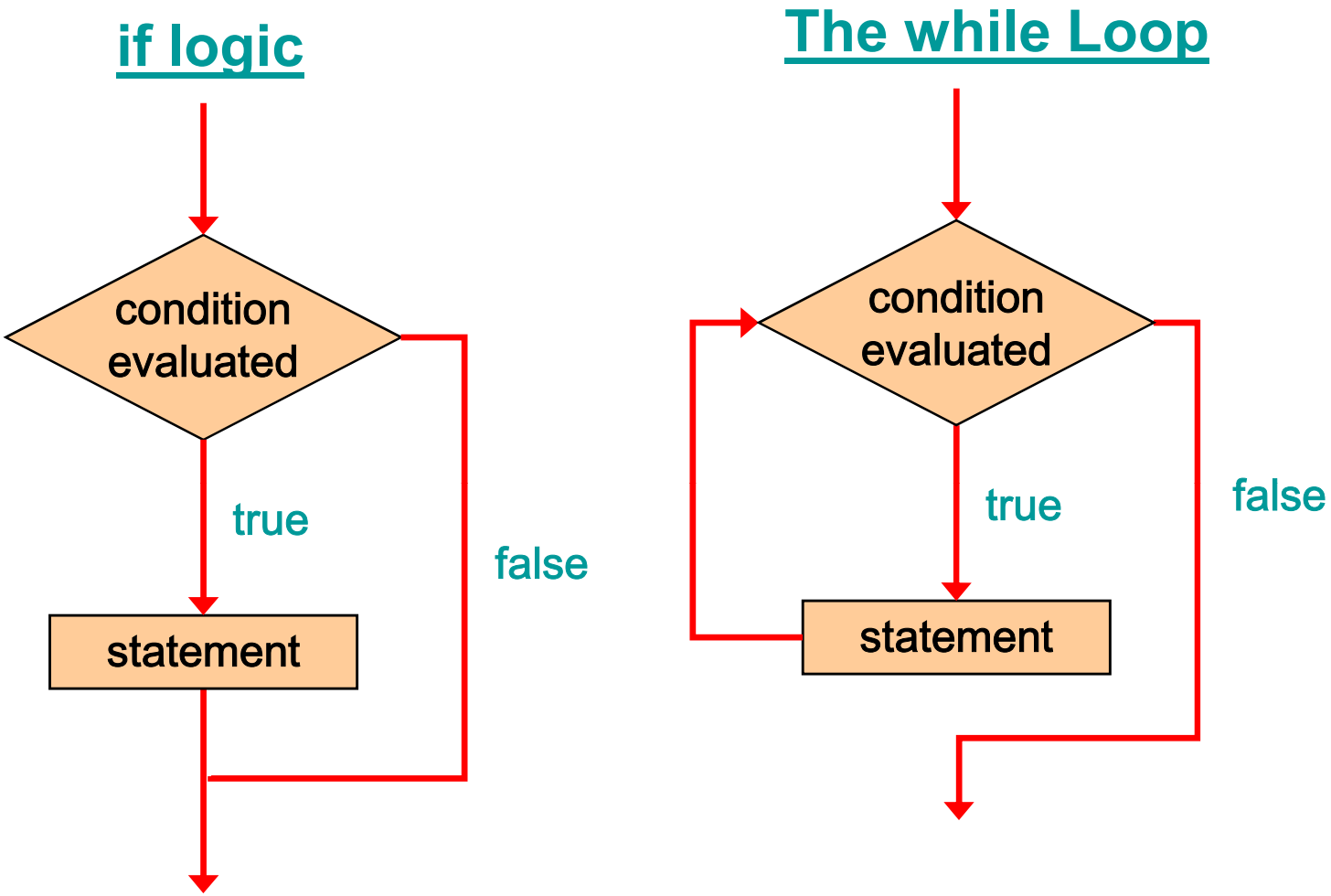
যতক্ষন যদি

```
while ( condition )
    statement;
```

Logic of an if statement



Logic of a while Loop



The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

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

  - If the `condition` is true, the `statement or a block of statements` is executed
  - Then the condition is evaluated again, and if it is still true, the statement/block is executed again
  - The statement/block is executed repeatedly until the condition becomes false

The while Statement

- **An example of a while statement:**

```
int count = 1;
while (count <= 5)
{
   printf ("%d\n", count);
   count++;
}
```

  - **If the condition of a `while` loop is false initially, the statement is never executed**
  - **Therefore, the body of a `while` loop will execute zero or more times**

The while Statement

- **Example program that continuously asks for positive number as input:**

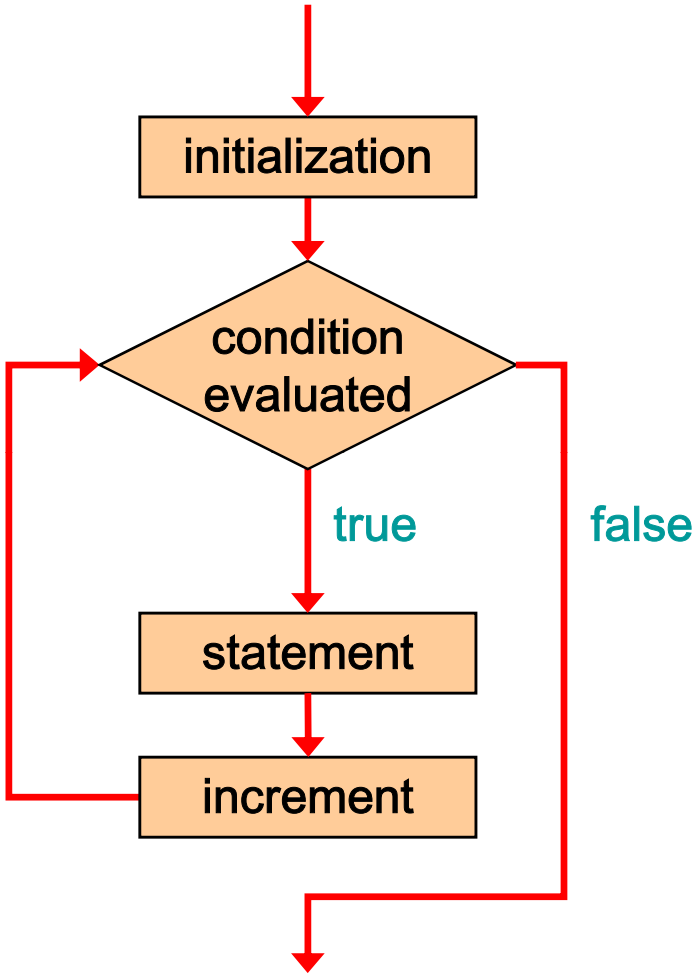
```
int n;

printf (“Please enter a positive number:”);
scanf(“%d”,&n);
while (n < 0){
  printf (“Enter positive number, BE POSITIVE!\n”);
  scanf(“%d”, &n);
}
```

Some examples

- Print “Sky is the limit!” 10 times.
- Print “Sky is the limit!” 100 times.
- Print “Sky is the limit!” n times.
- Print first n numbers.
- Print odd numbers up to n.
- Print even numbers up to n.
- Print summation of first n numbers.
- Print summation of all odd numbers up to n.
- Print summation of all even numbers up to n.
- Print factorial of n
- Print x^n , where x and n are integers.

Logic of a for loop



The for Statement

- **A *for statement* has the following syntax:**

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

The for Statement

- **An example of a `for` loop:**

```
for (int count=1; count <= 5; count++)
   printf ("%d\n", count);
```

  - **The initialization section can be used to declare a variable**
  - **Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body**
  - **Therefore, the body of a `for` loop will execute zero or more times**

The for Statement

- **The increment section can perform any calculation**

```
int num;
for (num=100; num > 0; num -= 5)
   printf ("%d\n", num);
```

  - **A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance**

Some example problems

- **Write down a program to find the summation of the following series:**

$$1^2 - 2^2 + 3^2 - 4^2 + ...\text{up to } n^2$$

- **Show all factors of a number n**
- **Show smallest factor of a number n (other than 1)**
- **Show largest factor of a number n (other than itself)**
- **Perfect number testing**
- **Prime number testing**
- **GCD of two numbers**
  - **Normal way**
  - **Efficient way**
- **Fibonacci number**

The for Statement

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

Infinite Loops

- **The body of a `while` loop eventually must make the condition false**
- **If not, it is called an *infinite loop*, which will execute until the user interrupts the program**
- **This is a common logical error**
- **You should always double check the logic of a program to ensure that your loops will terminate normally**

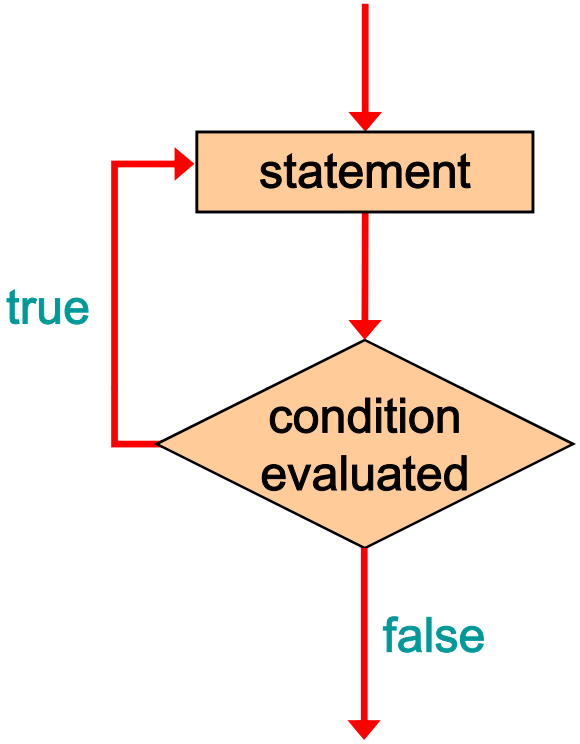
Infinite Loops

- **An example of an infinite loop:**

```
int count = 1;
while (count <= 25)
{
   printf ("%d\n", count);
   count = count - 1;
}
```

  - **This loop will continue executing until interrupted (Control-C) or until an underflow error occurs**

Logic of a do-while Loop



The do Statement

- A *do statement* has the following syntax:

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

  - The `statement` is executed once initially, and then the `condition` is evaluated
  - The statement is executed repeatedly until the condition becomes false

The do Statement

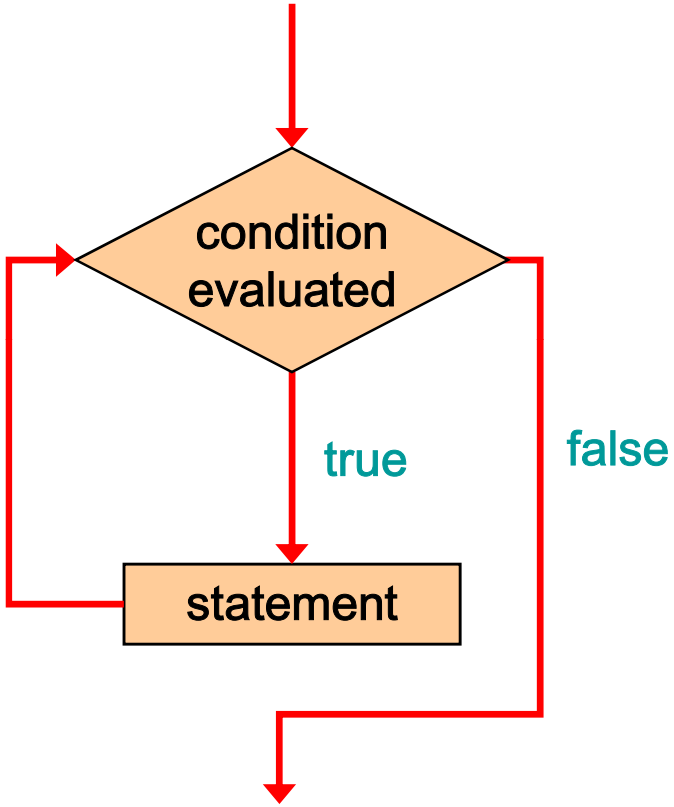
- **An example of a `do` loop:**

```
int count = 1;
do
{
   printf("%d\n", count);
   count++;
} while (count <= 5);
```

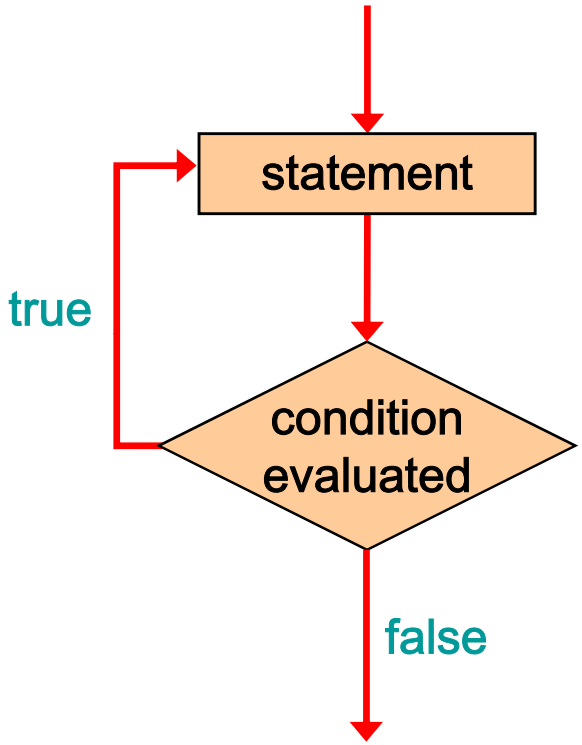
- **The body of a `do` loop is executed at least once**

Comparing while and do

The while Loop



The do Loop



Example: Printing reverse of a number

- **Write down a program that prints the digits of a number in reverse.**

- **For example:**
- **input: 6457**
- **output: 7546**

```
scanf("%d",&n);
do{
        a = n%10;
        printf("%d",a);
        n = n/10;
} while (n != 0);
```

Relevant Problem: counting number of digits of a number

- **Write down a program that prints number of digits of a number *n*.**
- **For example:**
- **input: 6457**
- **output: 4**

```
scanf("%d",&n);
C = 0;
do{
        n = n/10;
        c++;
} while (n != 0);
printf("%d",c);
```

Nested Loops

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops

- **What will the output?**

```
for(i=1; i <= 3; i++){
      for(j=1; j <= 2; j++){
            printf("Sky is the limit\n");
      }
      printf("The world is becoming smaller\n");
}
```

Output

```
for(i=1; i <= 3; i++){
        for(j=1; j <= 2; j++){
                printf("Sky is the limit\n");
        }
        printf("The world is becoming smaller\n");
}
```

```
Sky is the limit
Sky is the limit
The world is becoming smaller
Sky is the limit
Sky is the limit
The world is becoming smaller
Sky is the limit
Sky is the limit
The world is becoming smaller
```

Output

```
for(i=1; i <= 3; i++){
      for(j=1; j <= 2; j++){
            printf("%d %d\n",i,j);
      }
}
```

```
1 1
1 2
2 1
2 2
3 1
3 2
```

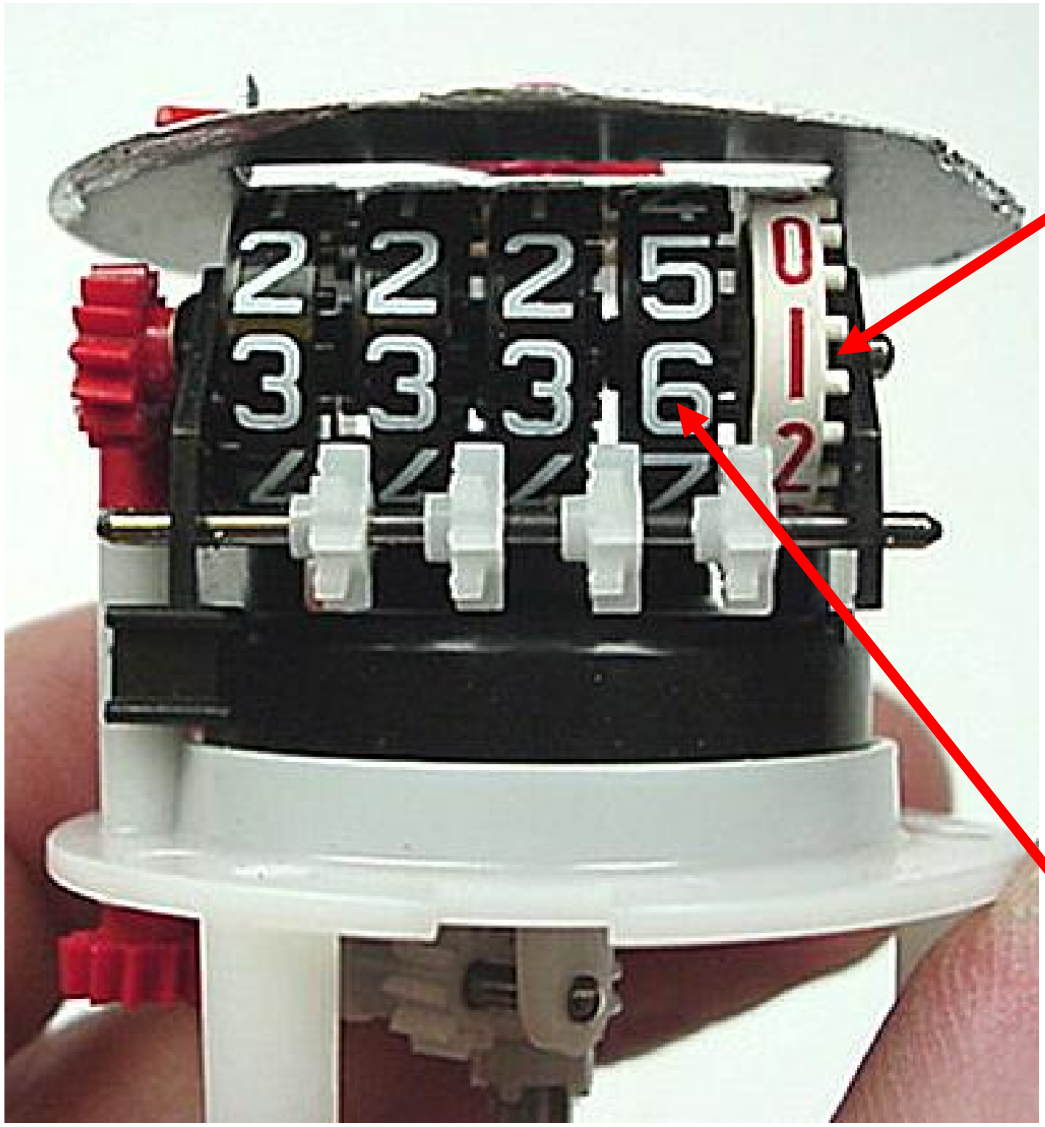
Nested Loops

- How many times will the string `"Here"` be printed?

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 <= 20)
   {
      printf ("Here \n");
      count2++;
   }
   count1++;
}
```

10 * 20 = 200

Analogy for Nested Loops



Inner Loop

Outer Loop

Example: Stars

- **Write a program that prints the following. The total number of lines will be input to your program.**

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Example: Stars

- **Write a program that prints the following. The total number of lines will be input to your program.**

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Some more Examples

- **Write a program that prints all prime numbers up to *x*. The integer x will be input to your program.**

- **Write a program that prints all prime factors of a number n given as input.**

- **Write a program that prints all perfect numbers up to *x*. The integer x will be input to your program.**