University
Center of Mila

Introduction to programming
1st year ST – ENG & LMD

Chapter 4: Conditional and Iterative Control Structures

II. Loop Statements

By

Dr. Farouk KECITA

Academic year : 2025/2026

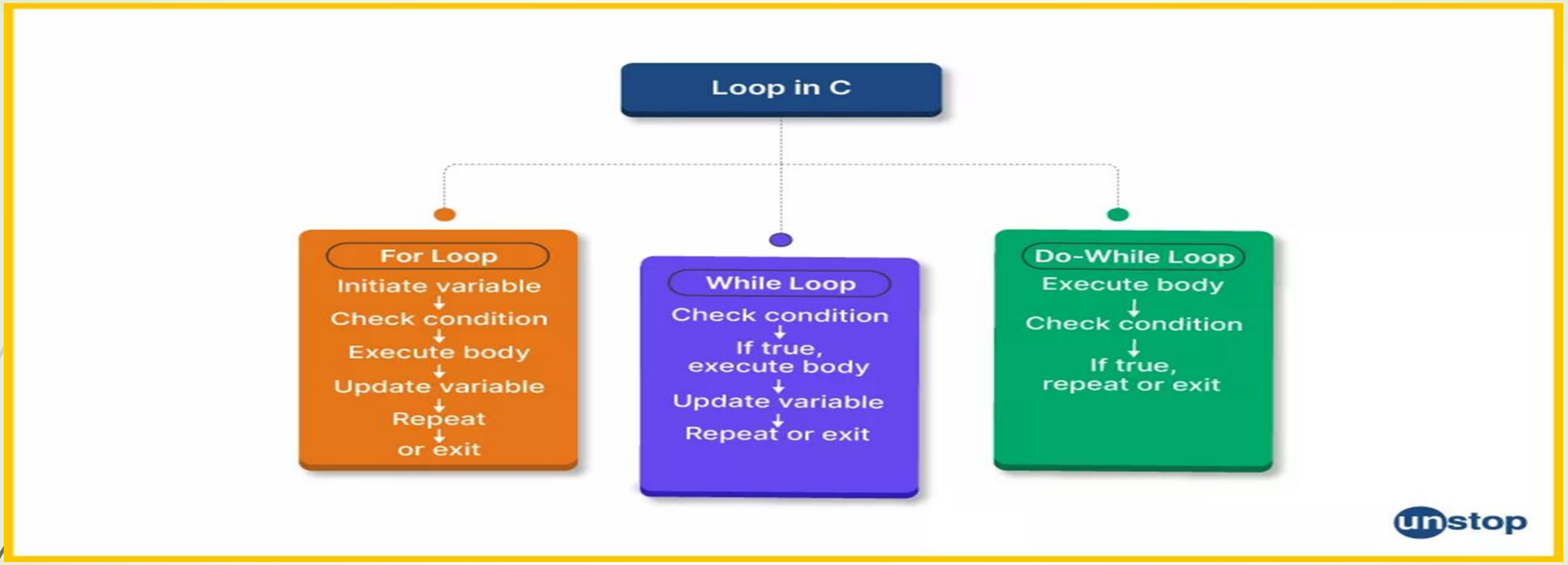
Introduction

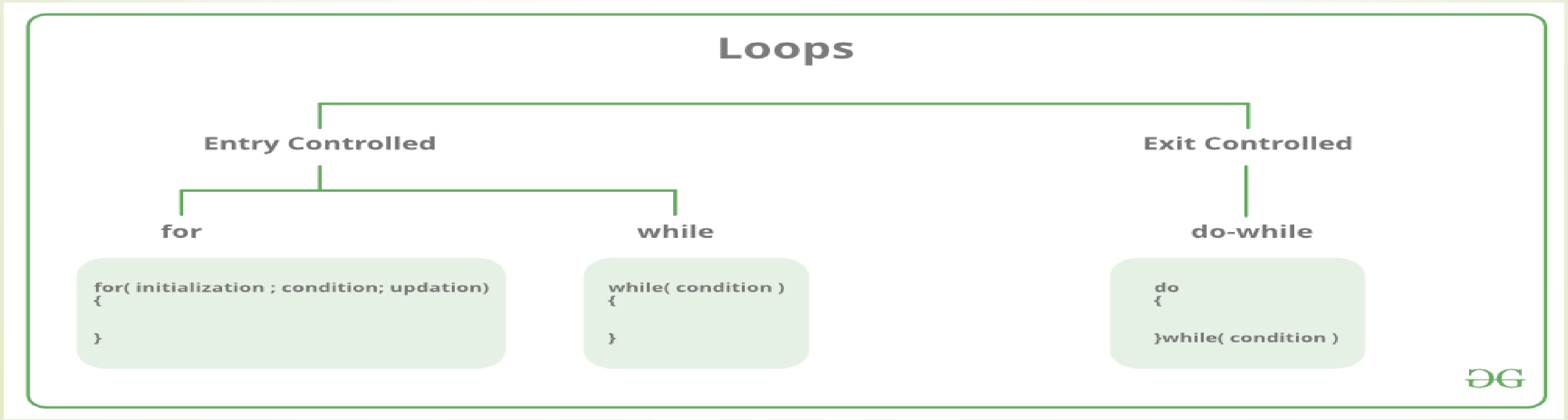
- Loop: it is a block of statement that performs set of instructions.
- In loops Repeating particular portion of the program either a specified number of time or until a particular no of condition is being satisfied.

❖ **There are mainly two types of loops in C Programming:**

- **Entry Controlled loops:** The test condition is checked before entering the main body of the loop. **For Loop and While Loop** is Entry-controlled loops.
- **Exit Controlled loops:** The test condition is evaluated at the end of the loop body. The loop body will execute at least once, irrespective of whether the condition is true or false. **do-while Loop** is Exit Controlled loop.

Introduction





```
for( initialization ; condition; updation)
{

}
```

```
while( condition )
{

}
```

```
do
{

}while( condition )
```

Introduction

Loop Type	Description
for loop	first Initializes, then condition check, then executes the body and at last, the update is done.
while loop	first Initializes, then condition checks, and then executes the body, and updating can be inside the body.
do-while loop	do-while first executes the body and then the condition check is done.

1. for Loop

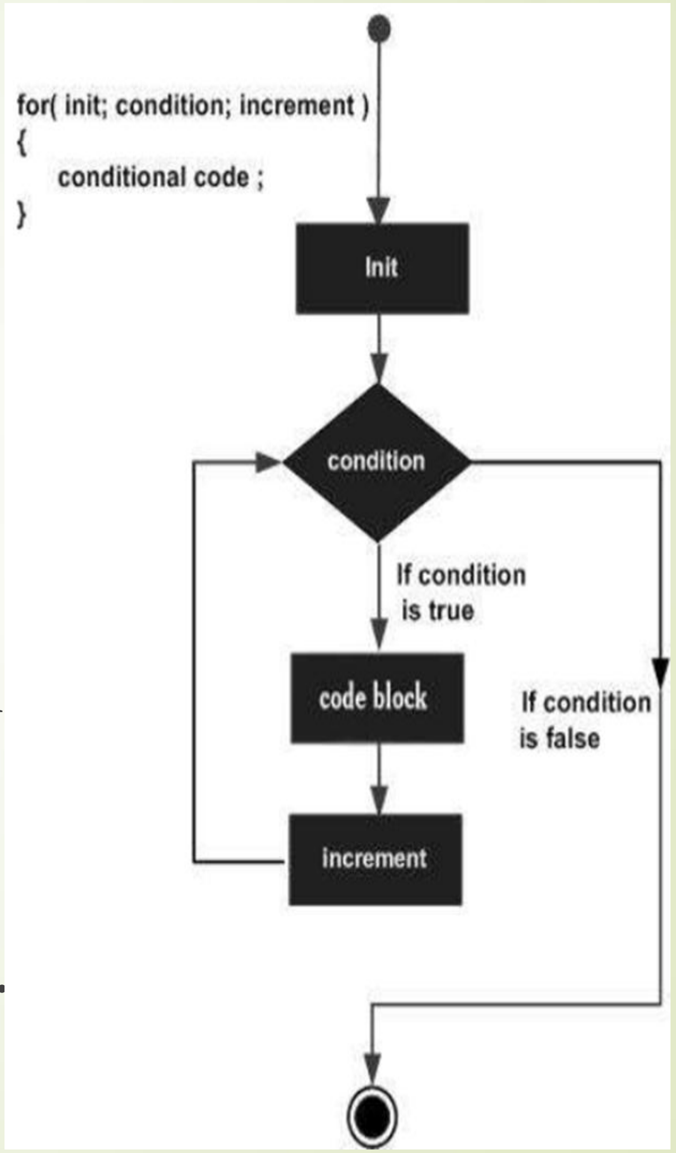
- **for loop** in C programming is a **repetition control** structure that allows programmers to write a loop that will be executed a **specific number of times**.
- **for loop** enables programmers to perform **N** number of **steps** together in a **single** line.

Syntax of for Loop in C:

```
for (initialize expression; test expression; update expression)
{
body of for loop;
}
```

1. for Loop

❖ **for loop Equivalent Flow Diagram:**

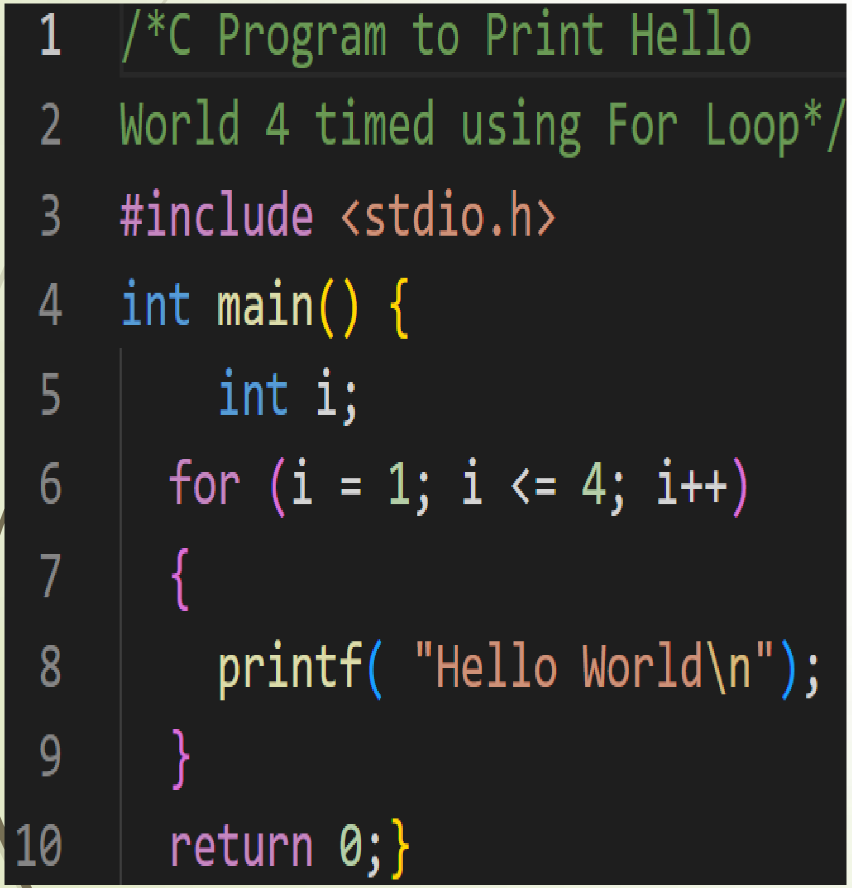


❖ **How for loop works?**

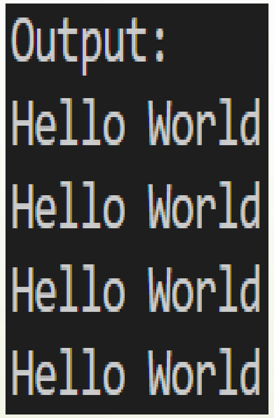
- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

1. for Loop

Example 1: C Program to Print “Hello World” 4 timed using For Loop

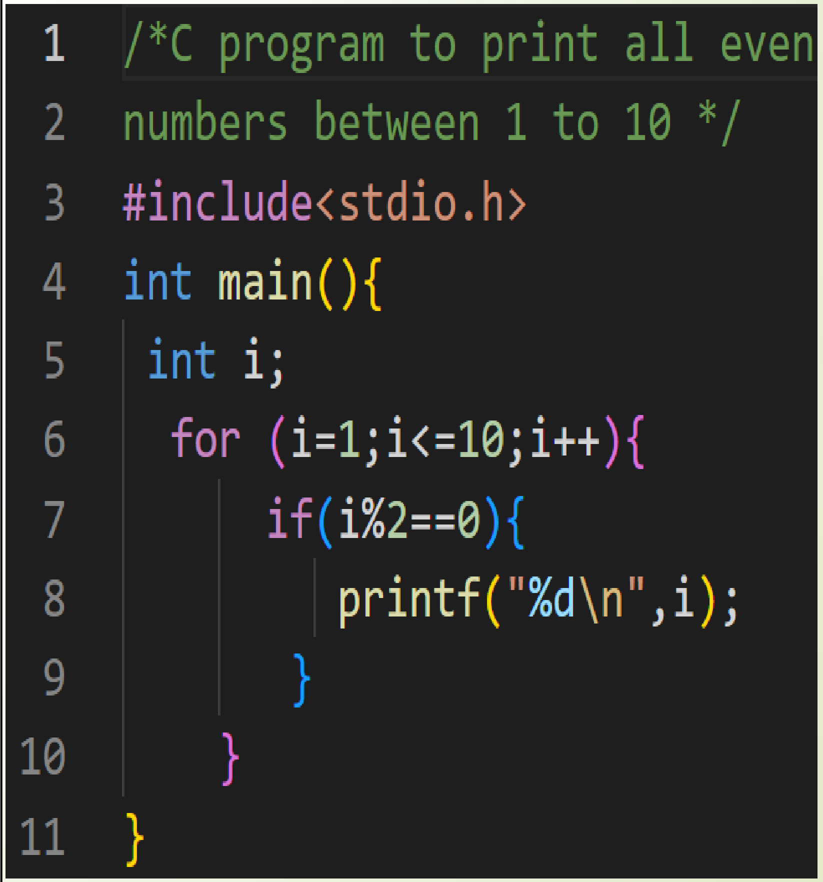


```c
/*C Program to Print Hello
World 4 timed using For Loop*/
#include <stdio.h>
int main() {
    int i;
  for (i = 1; i <= 4; i++)
  {
    printf( "Hello World\n");
  }
  return 0;}
```

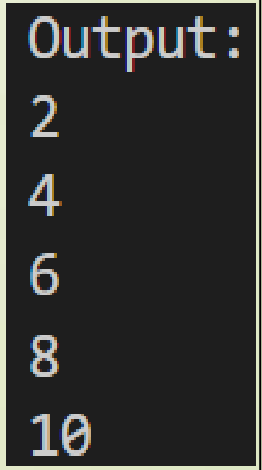


```
Output:
Hello World
Hello World
Hello World
Hello World
```

Example 2: Write a C program to print all even numbers between 1 to 10.



```c
/*C program to print all even
numbers between 1 to 10 */
#include<stdio.h>
int main(){
 int i;
  for (i=1;i<=10;i++){
      if(i%2==0){
         printf("%d\n",i);
      }
   }
}
```



```
Output:
2
4
6
8
10
```

2. while loop

- While loop does **not depend** upon the **number of iterations**.
- In for loop the number of iterations was previously **known** to us but in the While loop, the execution is terminated on **the basis** of the **test condition**.
- If the test condition will become **false** then it will **break** from the while loop **else body** will be **executed**.

❖ **Syntax of while Loop in C:**

```
initialization_expression;
while (test_expression)
{
body of the while loop;
update_expression;
}
```

2. while loop

- ❖ **while loop Equivalent Flow Diagram:**
- ❖ **How while loop works?**

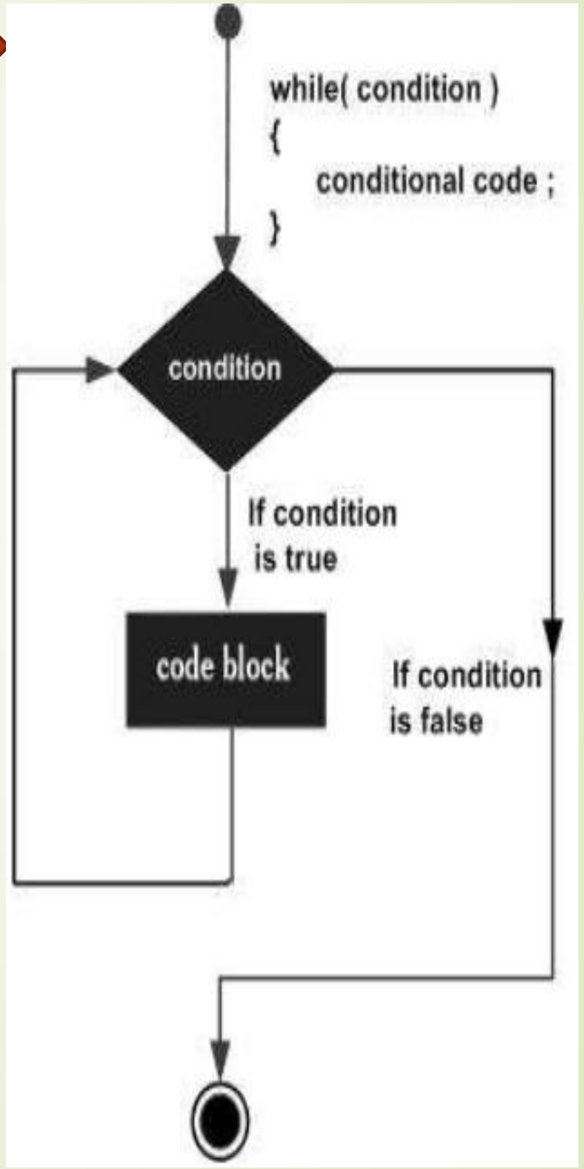
➢ Here, **statement(s)** may be a single statement or a block of statements.

➢ The **condition** may be any expression, and **true** is any nonzero value.

➢ The loop **iterates** while the condition is **true**.

➢ When the condition becomes **false**, the program control passes to the **statement immediately** following the loop.

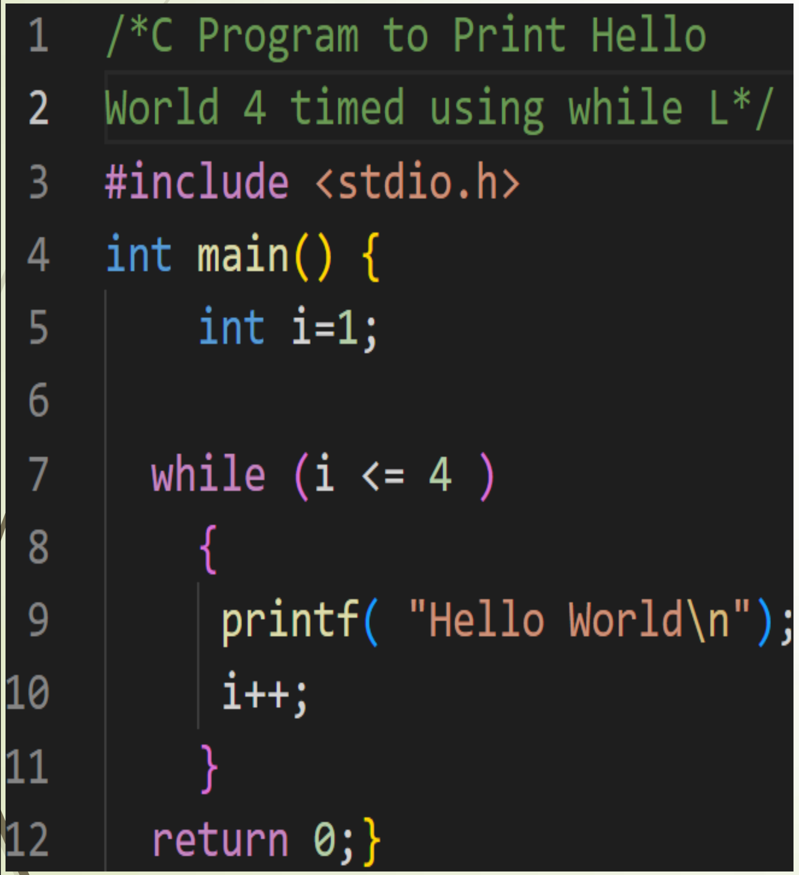
➢ The key point to note is that a **while loop** might **not execute at all**. When the condition is tested and the result is **false**, the loop body will be **skipped** and the first statement after the **while loop** will be **executed**.



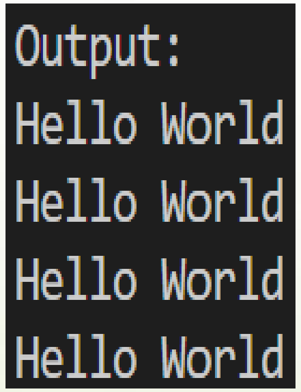
2. while loop

Example 1: C Program to Print “Hello World” 4 timed using while Loop

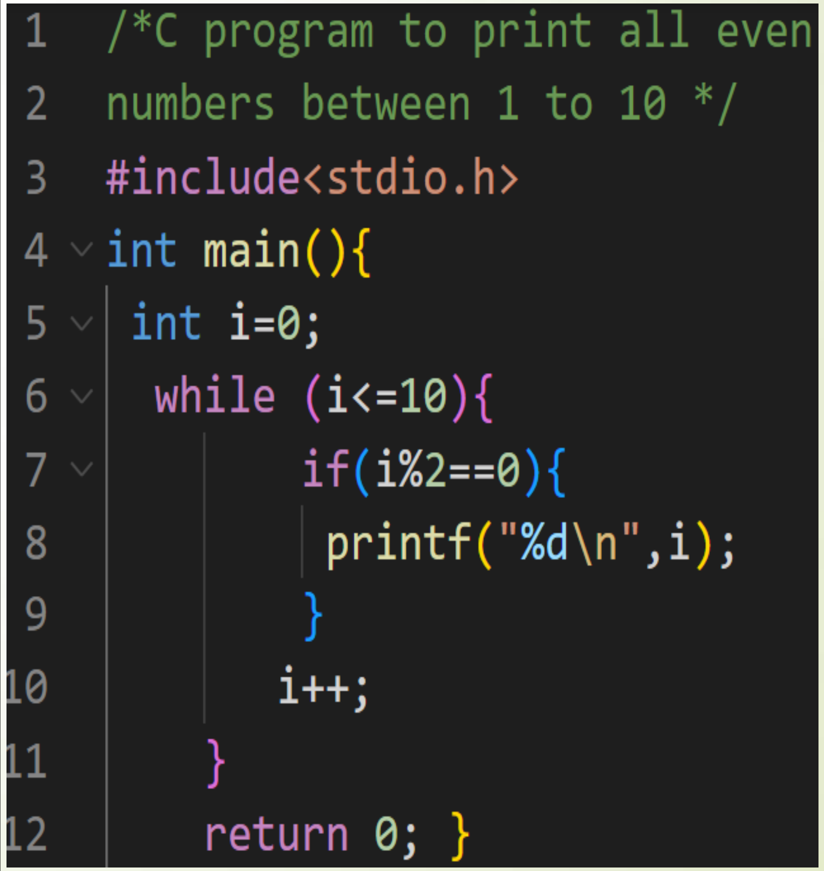
Example 2: Write a C program to print all even numbers between 1 to 10.



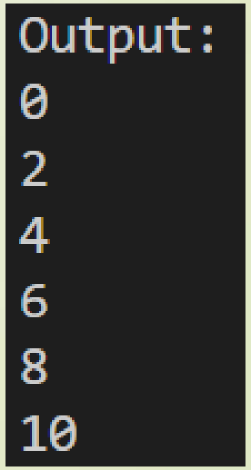
```c
/*C Program to Print Hello
World 4 timed using while L*/
#include <stdio.h>
int main() {
    int i=1;

  while (i <= 4 )
    {
     printf( "Hello World\n");
     i++;
    }
  return 0;}
```



```
Output:
Hello World
Hello World
Hello World
Hello World
```



```c
/*C program to print all even
numbers between 1 to 10 */
#include<stdio.h>
int main(){
 int i=0;
  while (i<=10){
        if(i%2==0){
         printf("%d\n",i);
        }
      i++;
    }
    return 0; }
```



```
Output:
0
2
4
6
8
10
```

3. do…while Loop

- The do…while in C is a loop statement used to repeat some part of the code till the given **condition is fulfilled**.
- It is a form of an exit-controlled or post-tested loop where the **test condition** is checked **after executing** the body of the loop.
- Due to this, the statements in the **do…while** loop will always **be executed** at **least once** no matter what the condition is.
- When you need to execute statements **at least for once irrespective** of the **result of the condition** then you have to use **do...while** loop.
- Unlike **while** loop, in which condition is checked at the **top of the loop**; in **do...while**, condition is checked **at the bottom**.

3. do…while Loop

❖ **Syntax of do…while Loop in C**

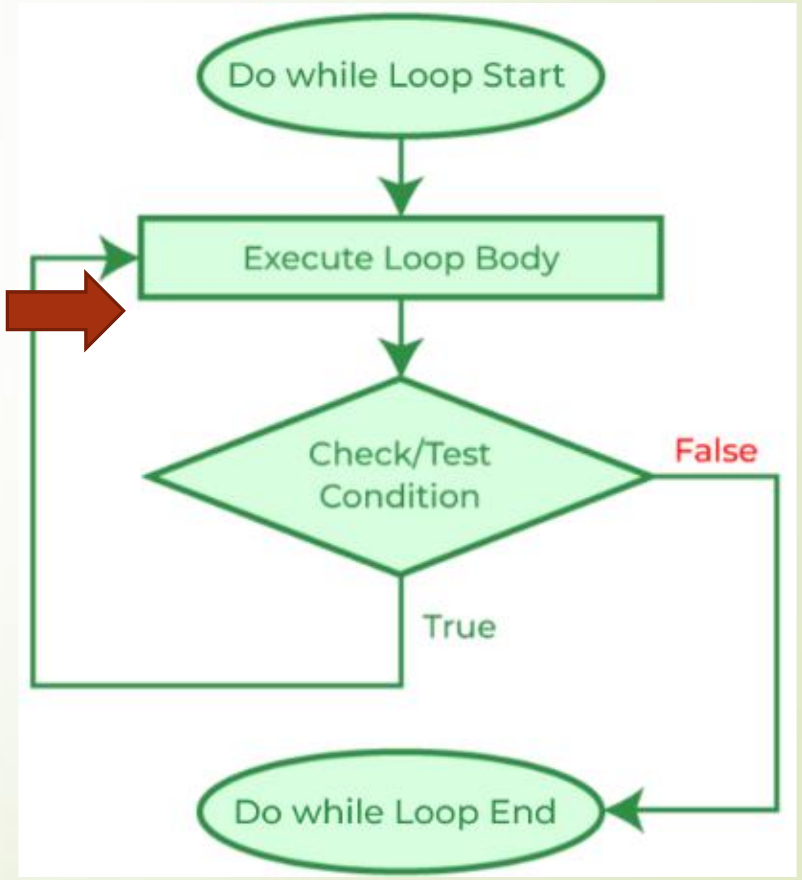
do {
body of do…while loop ;
}

while (*condition*);

❖ **while loop Equivalent Flow Diagram:**

❖ **How do … while loop works?**

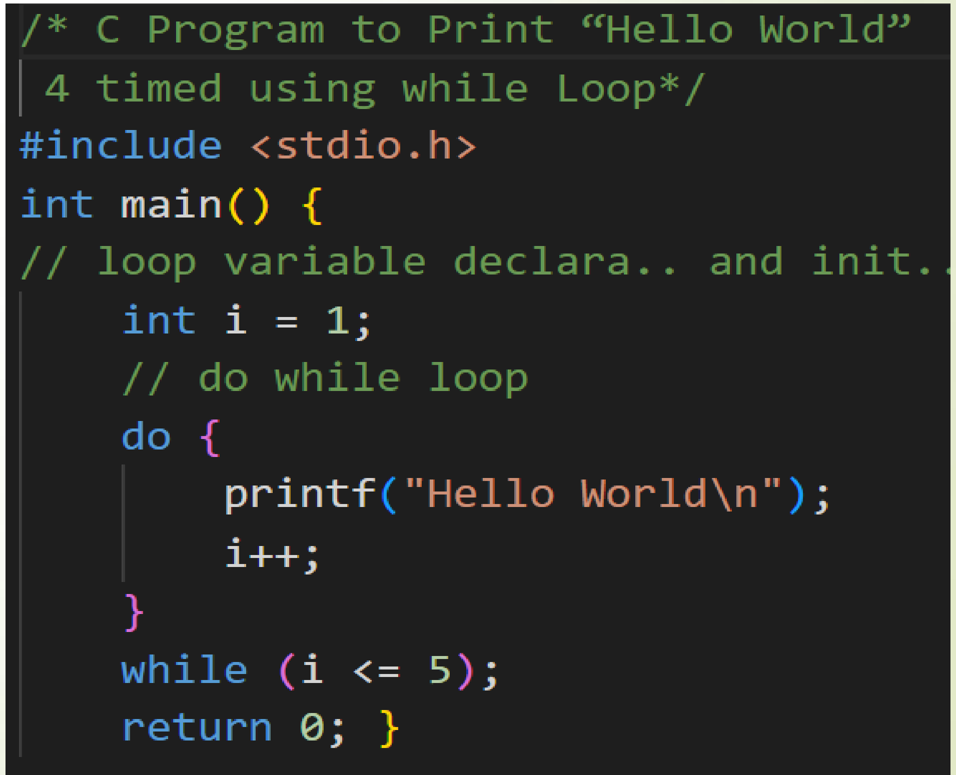
➢ When the program control first comes to the do…while loop, the body of the loop is executed first and then the test condition/expression is checked.



3. do…while Loop

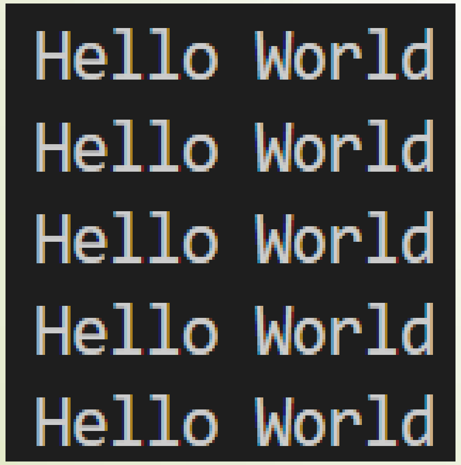
- When the test condition is evaluated as **true**, the program control goes to the **start** of the loop and the body is executed **once more**.
- The above process **repeats** till the test condition is **true**.
- When the test condition is evaluated as **false**, the program controls move on to the **next** statements after the do…while loop.

Example: Write a C Program to Print "Hello World" 5 timed using do…while Loop.



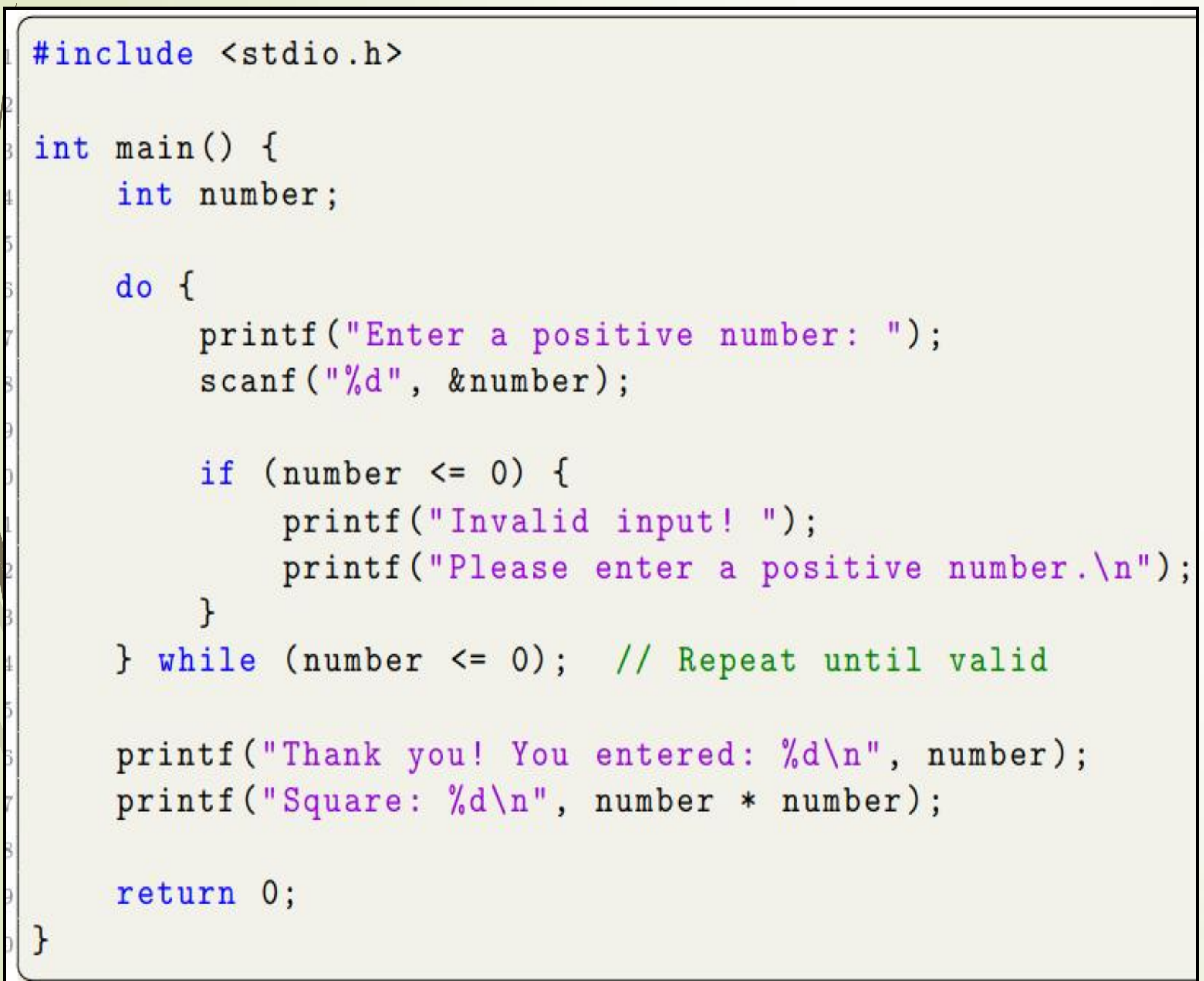
```
/* C Program to Print "Hello World"
 4 timed using while Loop*/
#include <stdio.h>
int main() {
// loop variable declara.. and init..
    int i = 1;
    // do while loop
    do {
        printf("Hello World\n");
        i++;
    }
    while (i <= 5);
    return 0; }
```

Out put:



```
Hello World
Hello World
Hello World
Hello World
Hello World
```

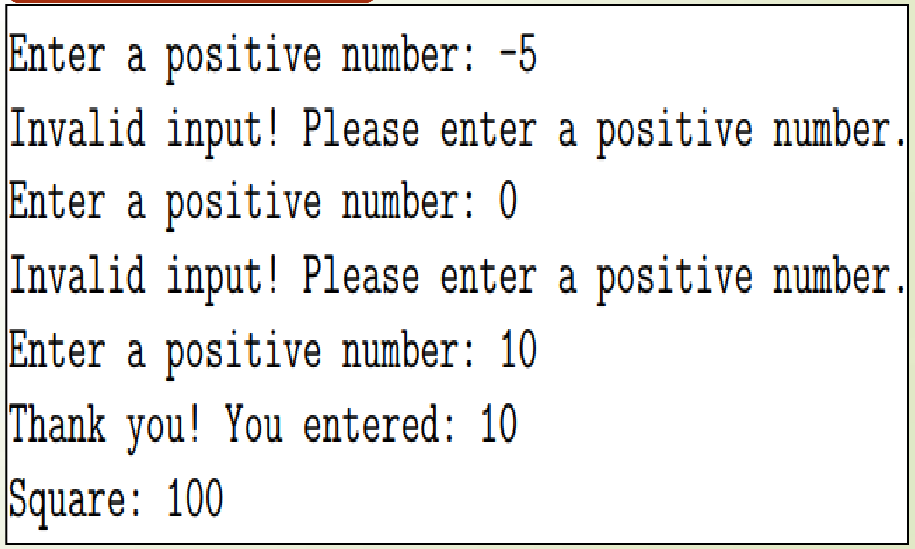
3. do…while Loop



```c
#include <stdio.h>

int main() {
    int number;

    do {
        printf("Enter a positive number: ");
        scanf("%d", &number);

        if (number <= 0) {
            printf("Invalid input! ");
            printf("Please enter a positive number.\n");
        }
    } while (number <= 0);  // Repeat until valid

    printf("Thank you! You entered: %d\n", number);
    printf("Square: %d\n", number * number);

    return 0;
}
```

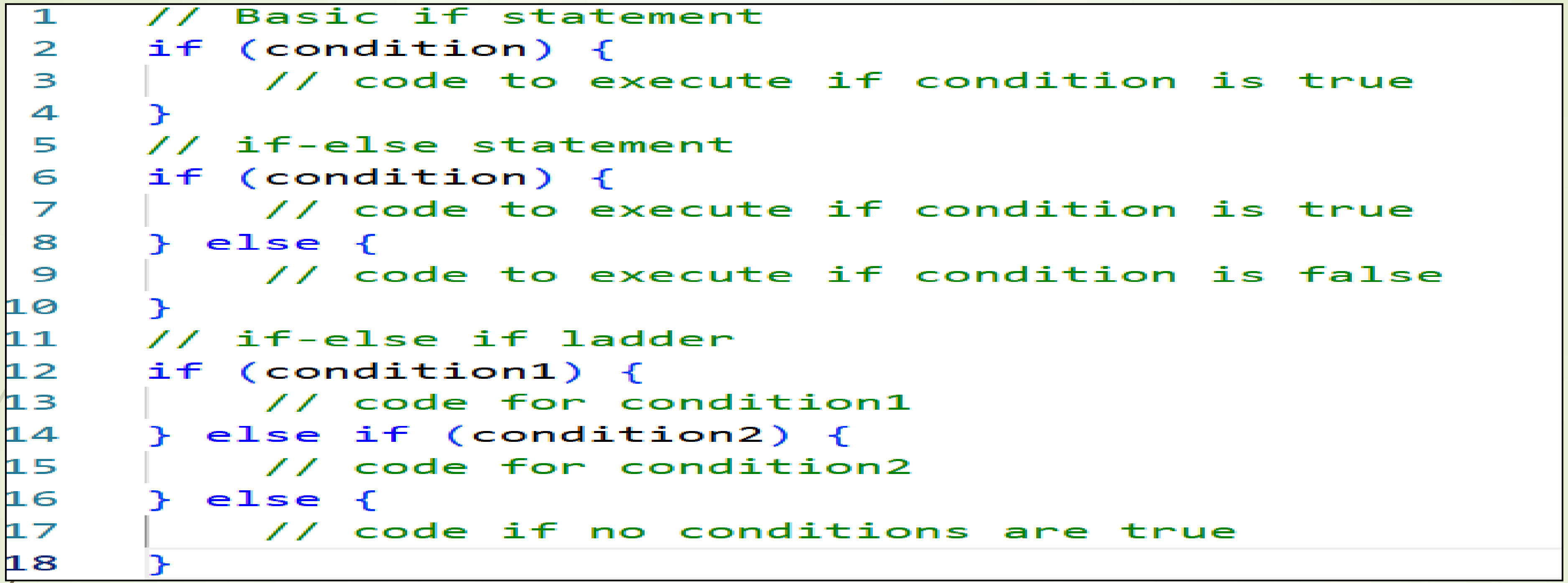
Example: *Write a C program that asks the user to enter a positive number. The program should continue asking for input until the user enters a positive number (greater than zero).*

Out put:

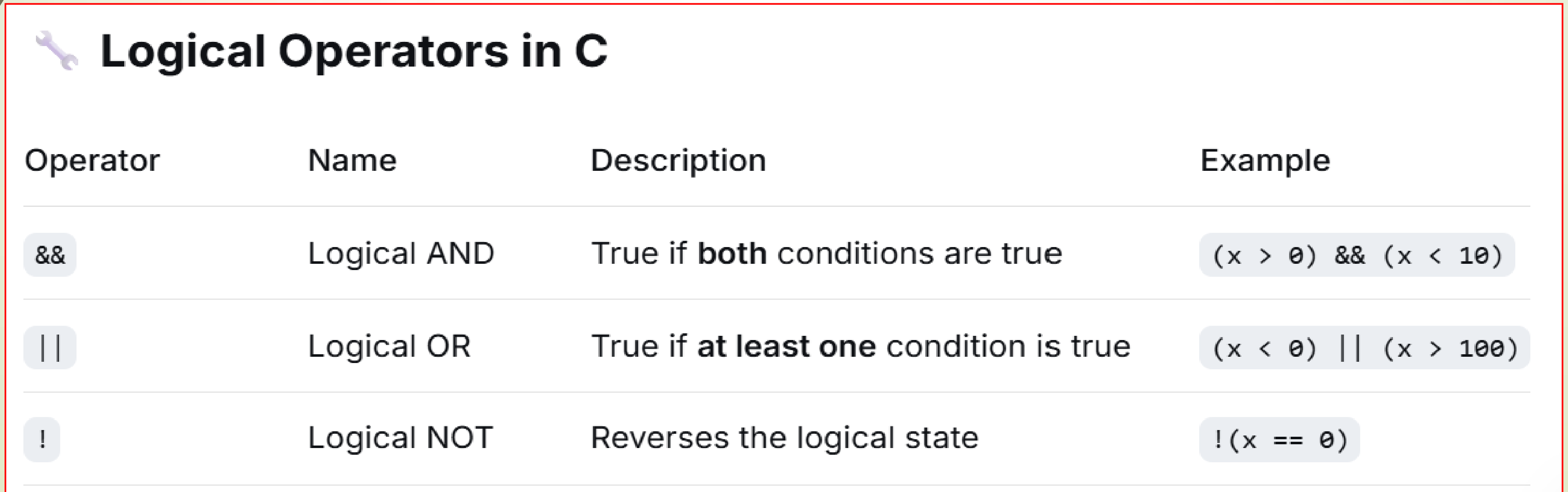


```
Enter a positive number: -5
Invalid input! Please enter a positive number.
Enter a positive number: 0
Invalid input! Please enter a positive number.
Enter a positive number: 10
Thank you! You entered: 10
Square: 100
```

4. Logical Operators and if Statement in C



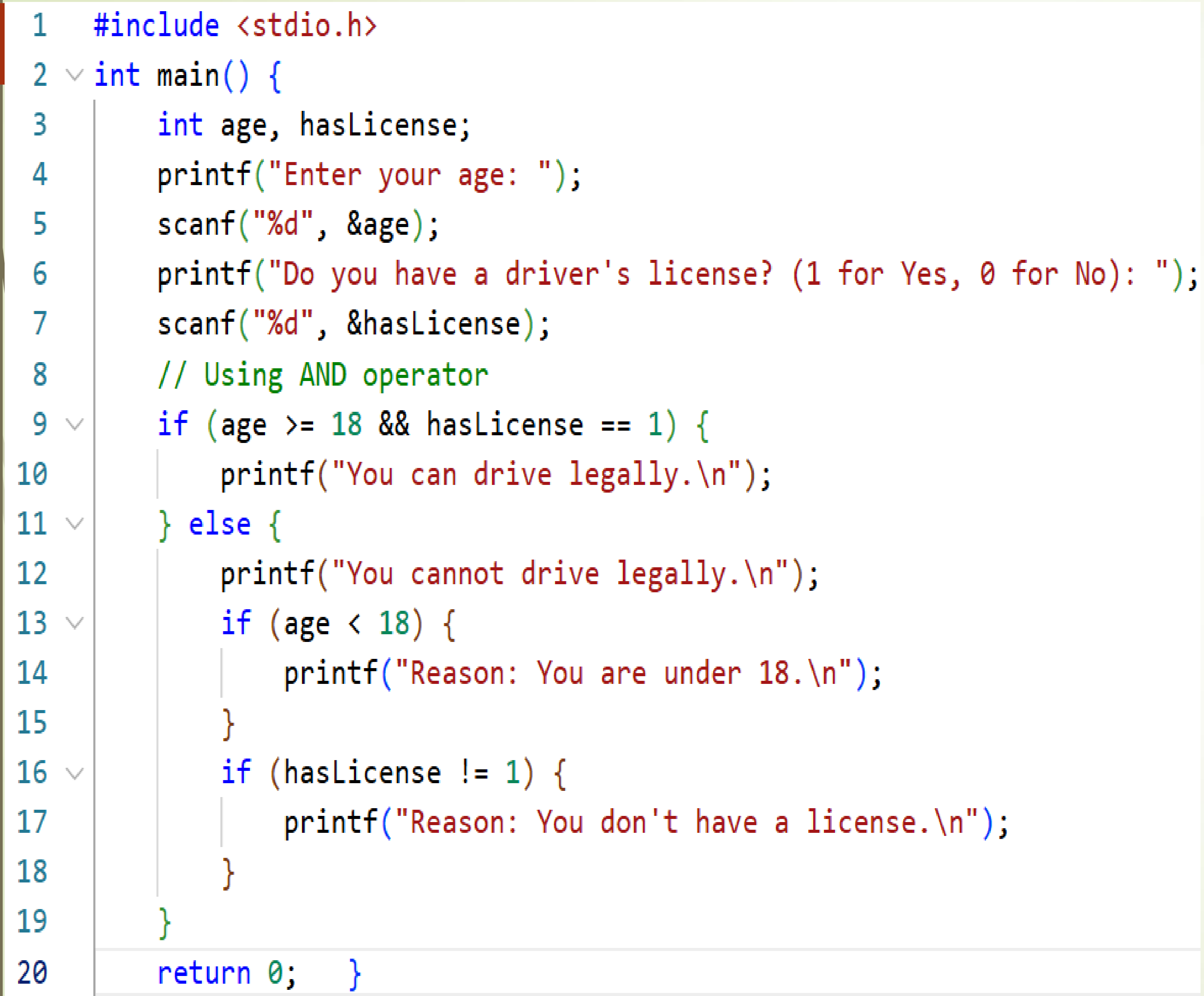
```c
// Basic if statement
if (condition) {
    // code to execute if condition is true
}
// if-else statement
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
// if-else if ladder
if (condition1) {
    // code for condition1
} else if (condition2) {
    // code for condition2
} else {
    // code if no conditions are true
}
```



Logical Operators in C

Operator	Name	Description	Example
`&&`	Logical AND	True if **both** conditions are true	`(x > 0) && (x < 10)`
`||`	Logical OR	True if **at least one** condition is true	`(x < 0) || (x > 100)`
`!`	Logical NOT	Reverses the logical state	`!(x == 0)`

4. Logical Operators and if Statement in C



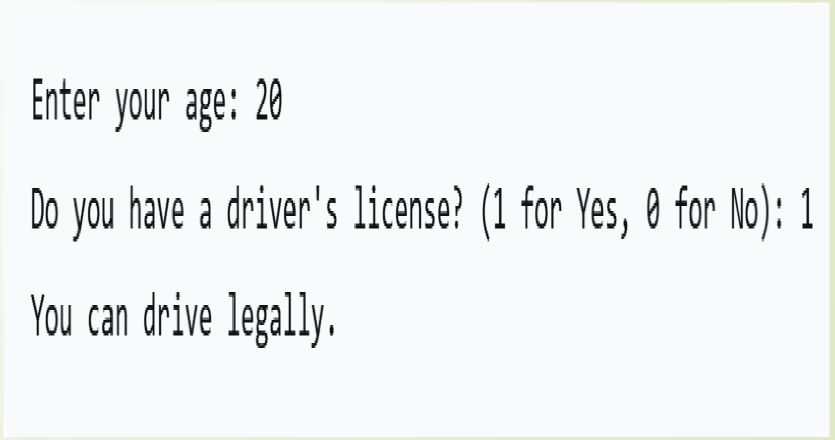
```c
#include <stdio.h>
int main() {
    int age, hasLicense;
    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Do you have a driver's license? (1 for Yes, 0 for No): ");
    scanf("%d", &hasLicense);
    // Using AND operator
    if (age >= 18 && hasLicense == 1) {
        printf("You can drive legally.\n");
    } else {
        printf("You cannot drive legally.\n");
        if (age < 18) {
            printf("Reason: You are under 18.\n");
        }
        if (hasLicense != 1) {
            printf("Reason: You don't have a license.\n");
        }
    }
    return 0;   }
```

Example 3: **Logical AND (&&):** ***Write a C program that checks if a person is eligible to drive legally based on two conditions:***

1. The person must be 18 years or older

2. The person must have a valid driver's license

Out put:



```
Enter your age: 20

Do you have a driver's license? (1 for Yes, 0 for No): 1

You can drive legally.
```

4. Logical Operators and if Statement in C

Example: **Logical OR (||)**

Write a C program that provides weather alerts based on temperature readings.

Program Requirements:

1. Ask the user to enter the current temperature in Celsius.
2. Use the logical OR operator (||) to check if the temperature represents extreme weather conditions:

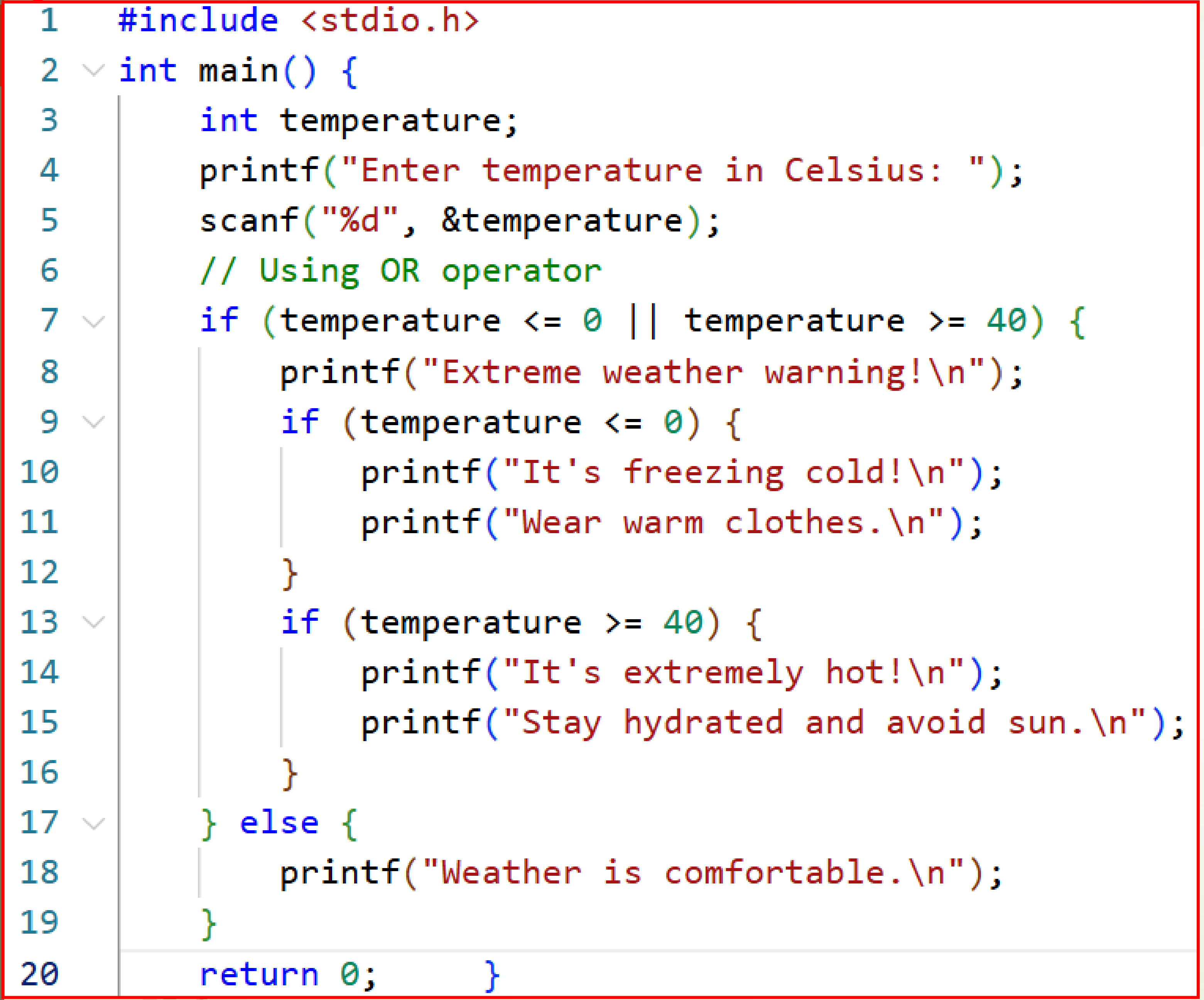
2_1. Temperature is 0°C or below (freezing)
2_2. OR temperature is 40°C or above (extremely hot)

3. If either extreme condition is true:

3_1. Display "Extreme weather warning!"
3_2. If temperature is ≤ 0°C, display additional messages about freezing conditions
3_3. If temperature is ≥ 40°C, display additional messages about extreme heat

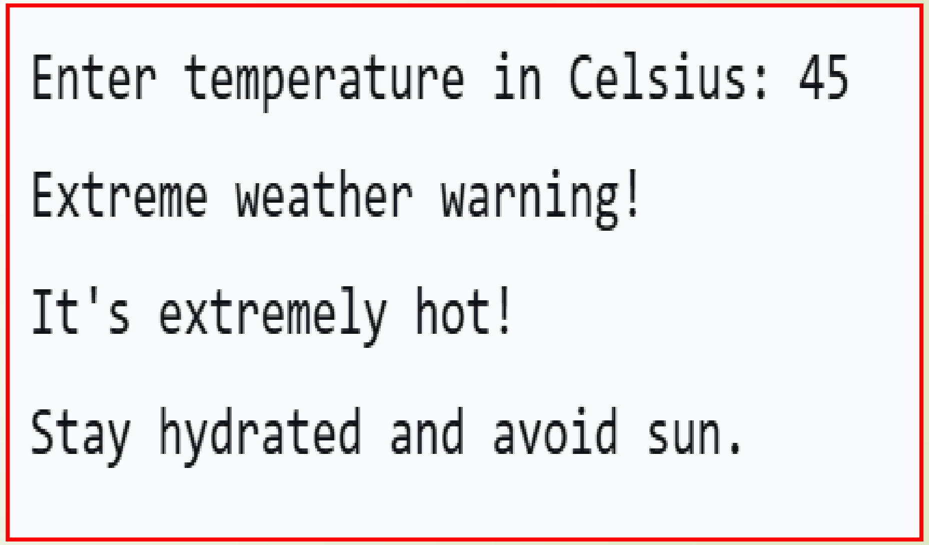
4. If neither extreme condition applies (temperature between 1°C and 39°C), display "Weather is comfortable."

4. Logical Operators and if Statement in



```c
#include <stdio.h>
int main() {
    int temperature;
    printf("Enter temperature in Celsius: ");
    scanf("%d", &temperature);
    // Using OR operator
    if (temperature <= 0 || temperature >= 40) {
        printf("Extreme weather warning!\n");
        if (temperature <= 0) {
            printf("It's freezing cold!\n");
            printf("Wear warm clothes.\n");
        }
        if (temperature >= 40) {
            printf("It's extremely hot!\n");
            printf("Stay hydrated and avoid sun.\n");
        }
    } else {
        printf("Weather is comfortable.\n");
    }
    return 0;     }
```

Out put:



```
Enter temperature in Celsius: 45
Extreme weather warning!
It's extremely hot!
Stay hydrated and avoid sun.
```