1. Introduction to Loop Interruptions

In C language, **loop interruption statements** are used to **change the normal execution flow of loops**. Normally, a loop runs until its condition becomes false. However, sometimes we need to:

- Stop a loop immediately
- Skip certain iterations
- Exit from nested loops
- Return control to another part of the program

Loop interruption statements provide this control.

---

2. Need for Loop Interruptions

Loop interruptions are required when:

- A specific condition is met before loop completion
- Unwanted iterations must be skipped
- User wants to terminate the loop early
- Error conditions occur during looping

Without loop interruption statements, programs may become inefficient or complex.

---

3. Types of Loop Interruption Statements in C

C language provides the following loop interruption statements:

1. **break**
2. **continue**
3. **goto**
4. **return**

Each statement has a specific purpose and usage.

---

4. break Statement

The **break statement** is used to **terminate the loop immediately** and transfer control to the statement following the loop.

Applicable in

- for loop
- while loop
- do–while loop
- switch statement

Syntax

```
break;
```

Working

- When break is executed, loop execution stops
- Control moves outside the loop

Example

```
int i;
for(i = 1; i <= 10; i++)
{
   if(i == 5)
      break;
   printf("%d ", i);
}
```

Output: 1 2 3 4

---

5. break in Nested Loops

In nested loops, **break terminates only the innermost loop**.

Example

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

---

6. continue Statement

The **continue statement** is used to **skip the remaining statements of the current iteration** and move to the next iteration of the loop.

Syntax

```
continue;
```

Working

- Skips code below it in the loop body
- Loop condition is checked again

Example

```
int i;
for(i = 1; i <= 5; i++)
{
   if(i == 3)
      continue;
   printf("%d ", i);
}
```

Output: 1 2 4 5

---

7. Difference Between break and continue

Feature	break	continue
Loop termination	Ends the loop	Skips iteration
Control flow	Moves outside loop	Moves to next iteration
Usage	Exit loop	Skip condition

---

8. goto Statement

The **goto statement** transfers control to a **labeled statement** within the same function.

Syntax

```
goto label;
...
label:
   statements;
```

Example

```
int i = 1;
start:
printf("%d ", i);
i++;
if(i <= 5)
   goto start;
```

---

9. Disadvantages of goto Statement

- Makes program difficult to understand
- Creates unstructured code
- Leads to "spaghetti code"
- Difficult debugging

 Hence, goto is discouraged in modern programming

---

10. return Statement in Loops

The **return statement** is used to **exit from a function**, even if it is inside a loop.

Syntax

```
return value;
```

Example

```
int check(int n)
{
   int i;
   for(i = 2; i < n; i++)
   {
      if(n % i == 0)
         return 0;
   }
   return 1;
}
```

---

11. Loop Control Flow Diagram Explanation

- Normal loop → executes till condition false
- break → exits loop immediately
- continue → jumps to next iteration
- return → exits function
- goto → jumps to labeled statement

---

12. Common Programs Using Loop Interruptions

12.1 Search an Element in Array

```
for(i = 0; i < n; i++)
{
   if(arr[i] == key)
```

```
  {
    printf("Found");
    break;
  }
}
```

12.2 Skip Even Numbers

```
for(i = 1; i <= 10; i++)
{
  if(i % 2 == 0)
    continue;
  printf("%d ", i);
}
```

13. Common Errors with Loop Interruptions

1. Misuse of break
2. Infinite loops due to continue
3. Excessive use of goto
4. Confusion in nested loops
5. Missing loop conditions

14. Advantages of Loop Interruption Statements

- Improve efficiency
- Reduce unnecessary iterations
- Simplify complex logic
- Enhance program control

15. Limitations of Loop Interruptions

- Overuse reduces readability
- goto makes code unstructured
- Improper usage leads to logical errors

16. Best Practices

- Use **break** only when necessary
- Prefer **continue** carefully

- Avoid **goto** whenever possible
- Use **return** logically

---

17. Conclusion

Loop interruption statements play an important role in controlling loop execution in C language. Proper use of **break**, **continue**, **goto**, and **return** helps in writing efficient and flexible programs. However, misuse can make programs complex and error-prone.