SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

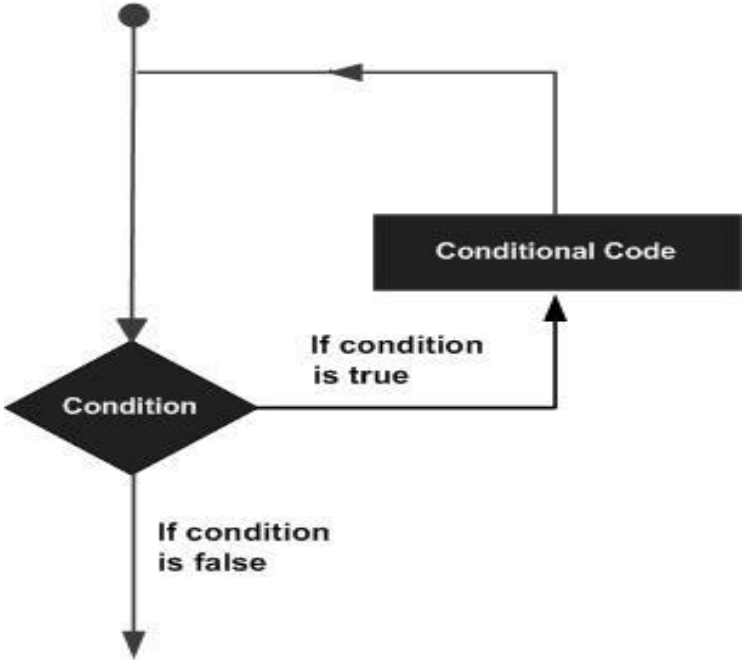
Unit II – C PROGRAMMING BASICS

Topic : Looping Statements - Illustrative Programs

LOOP

- A loop statement allows us to execute a statement or group of statements multiple times as long as a certain condition is true.

General Format



Types of Loop

- There are 3 types of Loop in C language, namely:
  - for loop
  - while loop
  - do while loop

for loop
(open ended / entry controlled loop)

- for loop is used to execute a set of statements repeatedly until a particular condition is satisfied.

```
for (initialization; condition; increment/decrement)
{
   statement_block;
}
```

nested for loop

- One for loop inside another for loop

```
for (initialization; condition; increment/decrement)
{
  for (initialization; condition; increment/decrement)
   {
     statement;
    }
}
```

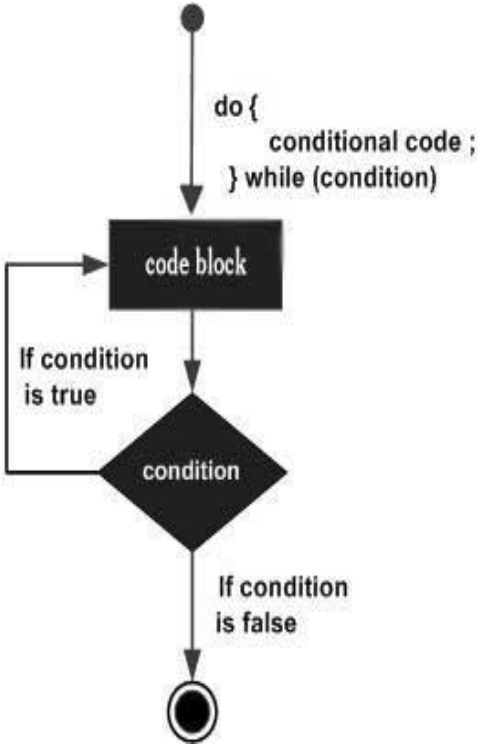
while Loop
(Entry controlled loop)

```
variable initialization;
while(condition)
{
    statements;
    variable increment or decrement;
}
```

do...while loop
(Exit controlled loop)

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

- If the condition is true, the flow of control jumps back, and the statement(s) in the loop executes again.
- This process repeats until the given condition becomes false.



Illustrative Programs:

Generate 10 natural numbers

Output
1 2 3 4 5 6 7 8 9 10

while loop

```
#include <stdio.h>
int main()
{
    printf("Using while loop:\n");
    int i = 1;
    while (i <= 10)
   {
        printf("%d ", i);
        i++;
    }
    printf("\n");
    return 0;
}
```

do...while loop

```
#include <stdio.h>
int main()
{
 printf("Using do-while loop:\n");
    int i = 1;
  do
    {
        printf("%d ", i);
        i++;
    }  while (i <= 10);
    printf("\n");
    return 0;
}
```

For loop

```
#include <stdio.h>
int main()
{
    printf("Using for loop:\n");

    for (int i = 1; i <= 10; i++)
   {
        printf("%d ", i);
    }
    printf("\n");
    return 0;
}
```

do...while loop

```
#include <stdio.h>
int main()
{
 printf("Using do-while loop:\n");
    int i = 1;
  do
    {
        printf("%d ", i);
        i++;
    }  while (i <= 10);
    printf("\n");
    return 0;
}
```

Example
Nested for loop

```
#include<stdio.h>
void main()
{
   int i,j;
   for(i=1 ; i < 5 ; i ++)
   {
      printf("\n");
      for(j= i ; j>0 ; j - -)
          printf("%d \t ", j);
    }
}
```

Output:

```
1
2       1
3       2       1
4       3       2       1
```

The Infinite Loop - for (;;)

- A loop becomes an infinite loop if a condition never becomes false.
- The **for** loop is traditionally used for this purpose.
- Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.
- When the conditional expression is absent, it is assumed to be true.

The Infinite Loop

```
#include <stdio.h>

int main() {
    printf("Infinite for loop:\n");
    for (;;)
   {   // No initialization, condition, or increment
        printf("This will run forever!\n");
    }
    return 0;
}
```

Comparison - while loop and do-while loop

Aspect	while Loop	do-while Loop
Condition Check	Before entering the loop (entry controlled loop /pre -test loop)	After the loop body (exit controlled loop / post-test loop)
Execution Guarantee	May not execute if the condition is false initially	Executes at least once, even if the condition is false initially
Loop Entry	Loops only if the condition is true from the start	Loops at least once, regardless of condition
Syntax	while (condition) { // code }	do { // code } while (condition);
Typical Use Case	When you want to check the condition before execution	When you want the loop to execute at least once
Example	while (x < 5) { printf("%d", x); x++; }	do { printf("%d", x); x++; } while (x < 5);
Behavior if Condition is False Initially	The loop may not run at all if the condition is false	The loop will execute at least once, even if the condition is false

Comparison - for and while loop

Aspect	for Loop	while Loop
Purpose	Typically used when the number of iterations is known beforehand.	Typically used when the number of iterations is not known.
Condition Check	Condition is checked at the start of each iteration.	Condition is checked before each iteration.
Initialization	Initialization is done in the loop itself.	Initialization must be done before the loop starts.
Update	Update (e.g., increment) is done in the loop itself.	Update must be handled inside the loop body.
Syntax	for (initialization; condition; update) { // code }	while (condition) { // code }
Typical Use Case	When the number of iterations is known in advance (e.g., for counting).	When the number of iterations is not known, and the loop depends on a condition.
Example	for (int i = 0; i < 5; i++) { printf("%d", i); }	int i = 0; while (i < 5) { printf("%d", i); i++; }

Unconditional control statements

Unconditional control statements can be powerful but should be used judiciously to maintain code readability.

goto:

Use only when absolutely necessary to avoid unstructured code.

break:

Used to exit loops or switch cases.

continue:

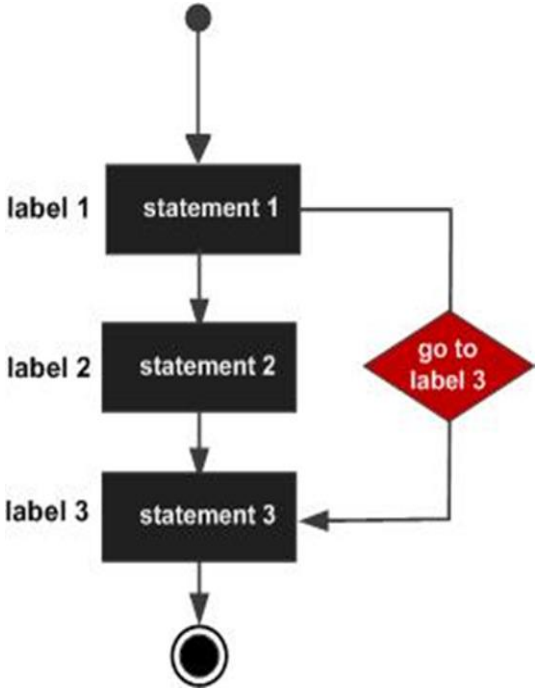
Skips the rest of the loop body for the current iteration.

return:

Exits a function, optionally returning a value.

goto statement

- The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement.
- The goto statement can be used to jump from anywhere to anywhere within a function.



THE GOTO STATEMENT

- The ‘goto’ statement **helps to branch unconditionally from one point to another in the program.**
- The goto **requires a label** in order **to identify the place where the branch is to be made.**
- A **label is any valid variable name** and must be **followed by a colon**.
- The **label is placed immediately before the statement** when the control is to be transferred.

THE GOTO STATEMENT
UNCONDITIONAL branching

Syntax:

```
goto label;
.
.
.
.
.
.
label:
  statement;
```

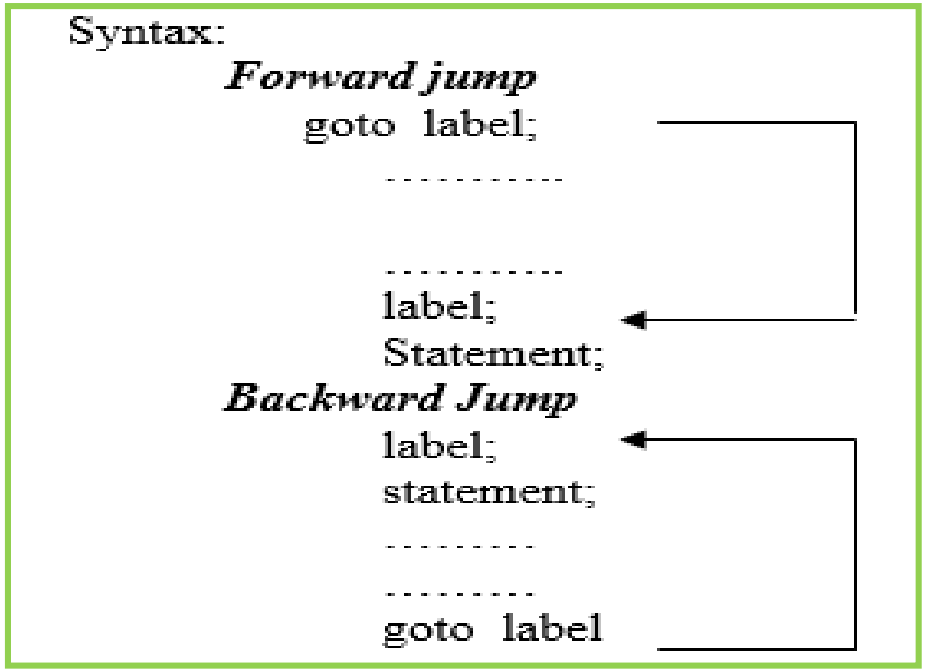
Syntax:

Forward jump

```
goto label;
----------
----------
label;
Statement;
```

Backward Jump

```
label;
statement;
---------
---------
goto label
```



THE GOTO STATEMENT

- The **label: can be anywhere in the program** before or after the goto label; statement.
- When a **goto statement is encountered**, the **flow of control will jump to the statement immediately** following the label in the goto statement.

THE GOTO STATEMENT

Backward jump:

- If the **label is before the statement goto label;** a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a backward jump.

Forward jump:

- If the **label is placed after the goto label;** some statements will be skipped and the jump is known as a forward jump.

THE GOTO STATEMENT

```
/*To print numbers from 1 to 10 using
goto statement*/
#include <stdio.h>
int main()
{
int number;
number=1;
repeat:
printf("%d\n",number);
number++;
if(number<=10)
goto repeat;
return 0;
}
```

Output

1
2
3
4
5

```
#include <stdio.h>
int main() {
    int x = 5;
    // Check condition
    if (x == 5) {
 goto forward_jump;  // Jump forward to the 'forward_jump' label
    }
    // This code will be skipped due to the forward jump
    printf("This will be skipped.\n");
    forward_jump:  // Label to jump to
    printf("This message is printed after the forward jump.\n");
    return 0;
}
```

Output : This message is printed after the forward jump.

THE GOTO STATEMENT

```
#include <stdio.h>

int main() {
    int i = 5;  // Start with 5 (first number in reverse order)

    backward_jump:  // Label to jump to
    if (i >= 1) {  // Check if i is greater than or equal to 1
        printf("%d\n", i);  // Print the current number
        i--;  // Decrement i to the next number in reverse order
        goto backward_jump;  // Jump back to backward_jump if condition is
true
    }
    return 0;
}
```

Output

5
4
3
2
1

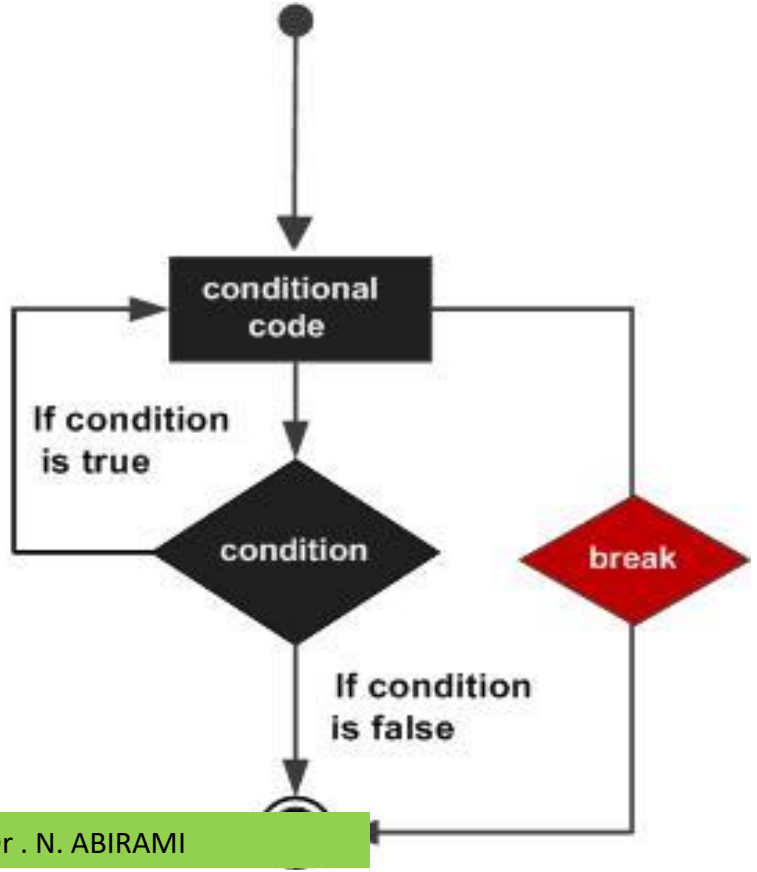
break statement

- The **break** statement in C programming has the following two usages
  - When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the **switch** statement (covered in the next chapter).

break statement

- Syntax

```
break;
```



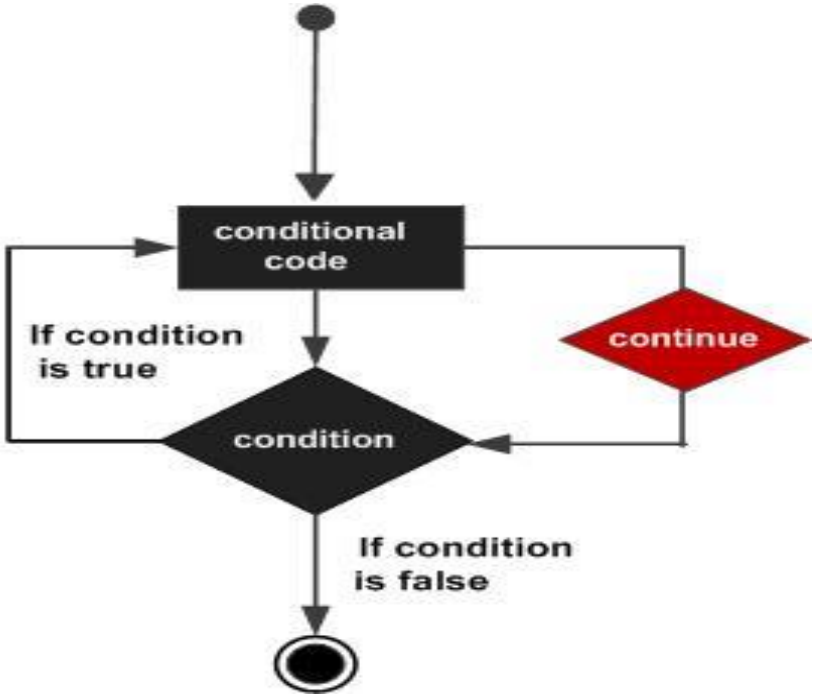
continue

- Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

continue

- Syntax

continue;



break statement

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 10; i++)
{
        if (i == 5)
   {    break;
// Exit the loop when i equals 5
   }
        printf("%d ", i);
    }
    return 0;
}
```

Continue statement

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 10; i++) {
       if (i % 2 == 0)
     {  continue;
       // Skip even numbers
       }
       printf("%d ", i);
   }
   return 0;
}
```

goto statement

```
#include <stdio.h>
int main() {
    int x = 0;
printf("Before the label.\n");
    goto skip;
 // Jump to the label "skip"
 printf("This will be skipped.\n");
skip:
    printf("After the label.\n");
    return 0;
}
```

return statement

```
#include <stdio.h>
int add(int a, int b)
{
    return a + b; // Return the sum
}
int main()
{
int result = add(3, 7);
printf("The sum is: %d\n", result);
    return 0;
}
```

Summary

S.No	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do --- while loop It is more like a while statement, except that it tests the condition at the end of the loop body.
4	nested loop can use one or more loops inside any other while, for, or do..while loop.


Thank
you