SNS COLLEGE OF TECHNOLOGY

Coimbatore-36.

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

COURSE NAME : 23CST101– PROBLEM SOLVING & C PROGRAMMING

I YEAR/ I SEMESTER

UNIT-II C PROGRAMMING BASICS

Topic: Branching and Looping Statements

Ms.Narmada C

AP/CSE

Department of Computer Science and Engineering

Branching Statements in C

The capacity to decide what to do and how to run distinct pieces of code based on specific circumstances is essential in the realm of programming. This ability to manage the execution flow is provided by ***branching statements*** in the C programming language.

Categories of branching statements

The various categories of **conditional branching** are as follows:

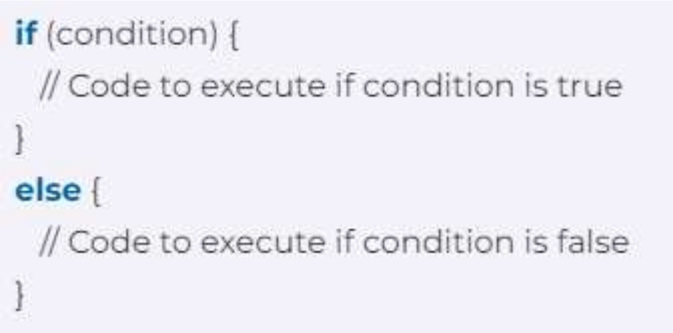
- **if Statement**
- **if-else Statement**
- **If-else ladder**
- **nested if-else Statement**
- **switch Statement**

The various categories of **Unconditional branching** are as follows:

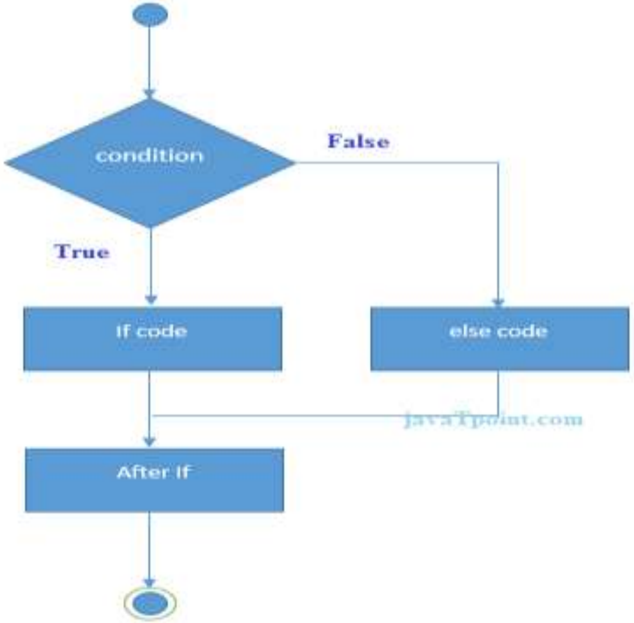
- **goto Statement**
- **break Statement**
- **continue Statement**
- **Return statement**

Conditional branching statements

if-else Statement



```
if (condition) {
  // Code to execute if condition is true
}
else {
  // Code to execute if condition is false
}
```



Program: Check if a number is odd or even.

```
#include <stdio.h>
int main()
{
int num = 4;
if (num % 2 == 0)
{
printf("Number is even.\n");
} else
{
printf("Number is odd.\n");
}
return 0;
}
```

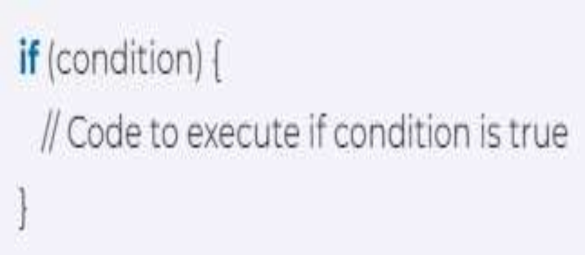
Input:

4

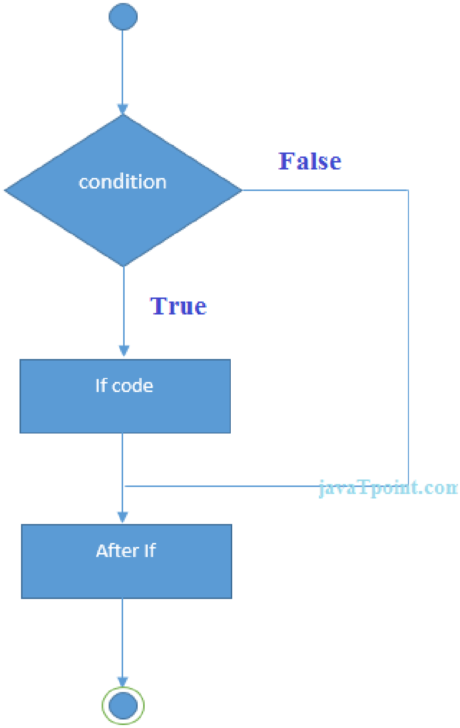
Output :

Number is even.

if Statement



```
if (condition) {
  // Code to execute if condition is true
}
```



Program: Check if a number is positive.

```
#include <stdio.h>
int main()
{
int num = 5;
(num > 0) { printf("Number is positive.\n");
}
return 0;
}
```

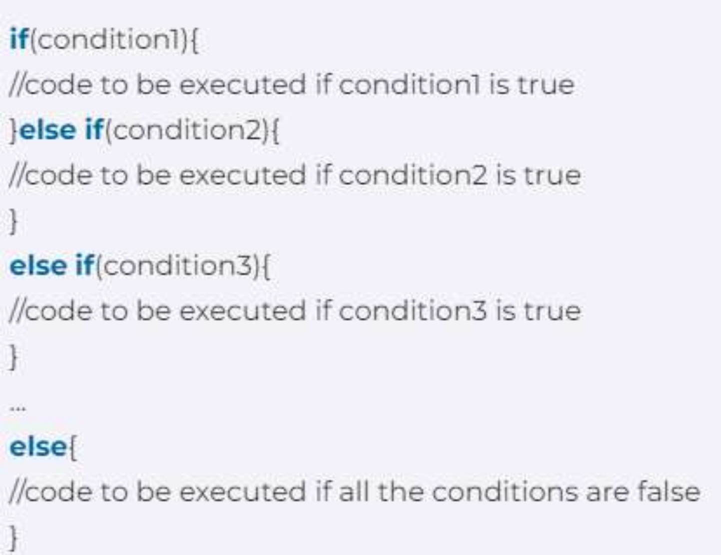
Input :

5

Output :

Number is positive.

if-else ladder Statement



```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

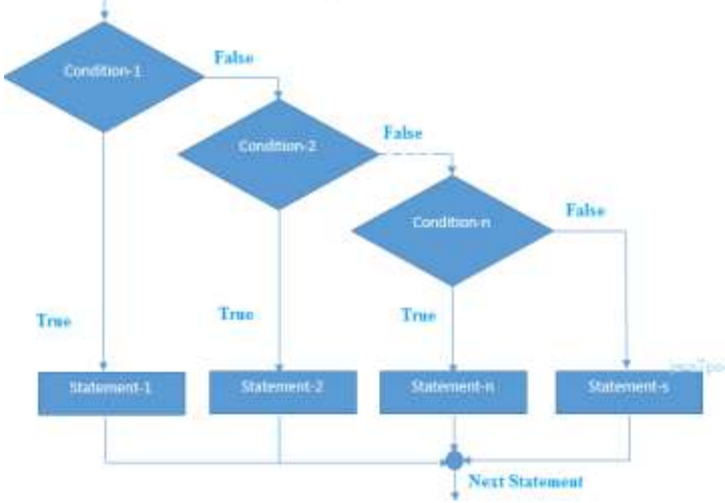
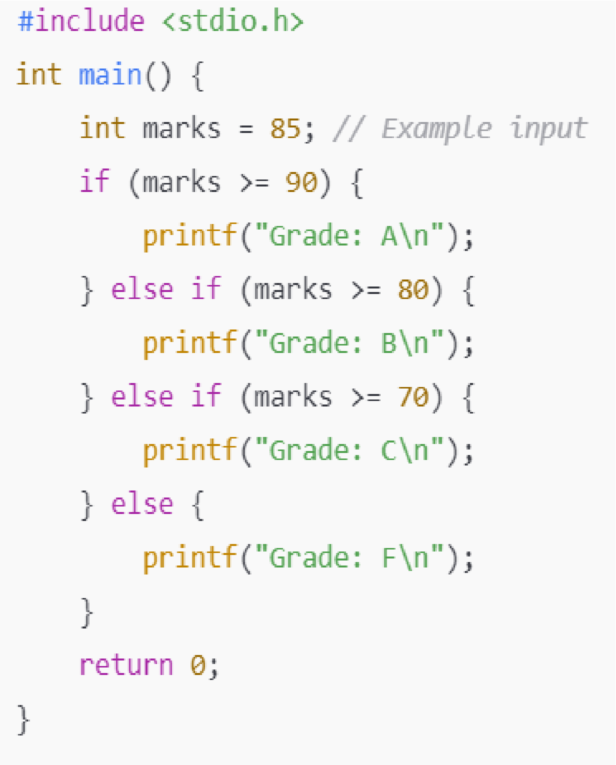


Fig: else-if ladder



```c
#include <stdio.h>
int main() {
    int marks = 85; // Example input
    if (marks >= 90) {
        printf("Grade: A\n");
    } else if (marks >= 80) {
        printf("Grade: B\n");
    } else if (marks >= 70) {
        printf("Grade: C\n");
    } else {
        printf("Grade: F\n");
    }
    return 0;
}
```

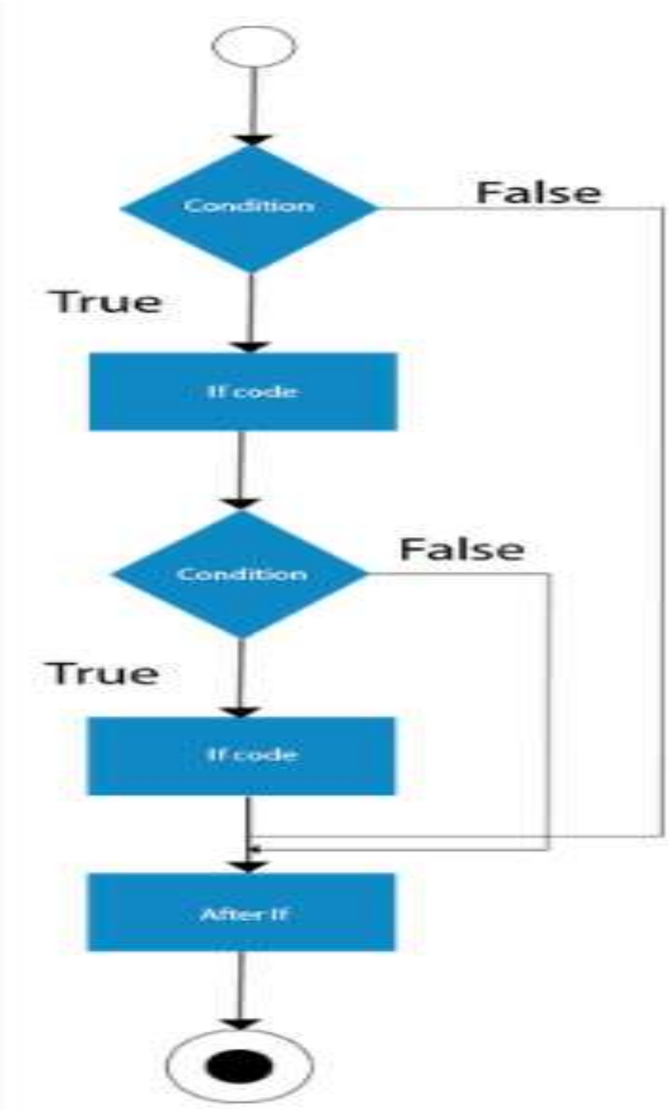
Input :

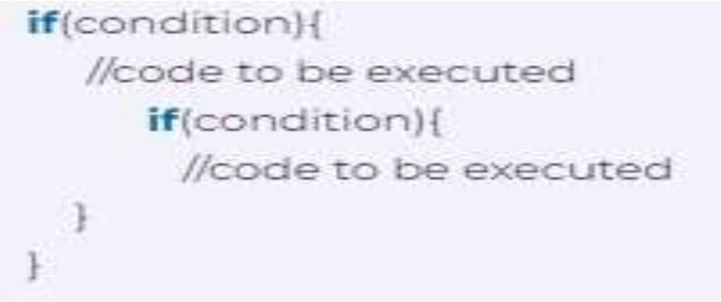
85

Output:

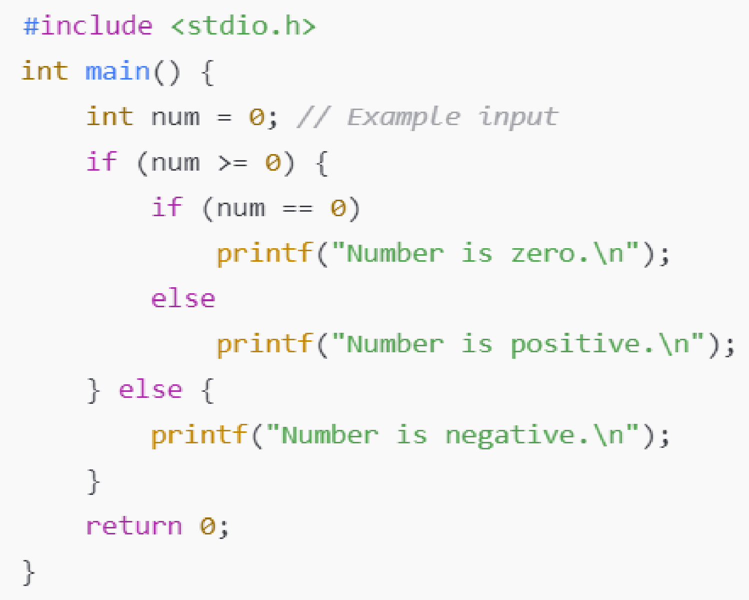
Grade: B

Nested if-else Statement





```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```



```
#include <stdio.h>
int main() {
    int num = 0; // Example input
    if (num >= 0) {
        if (num == 0)
            printf("Number is zero.\n");
        else
            printf("Number is positive.\n");
    } else {
        printf("Number is negative.\n");
    }
    return 0;
}
```

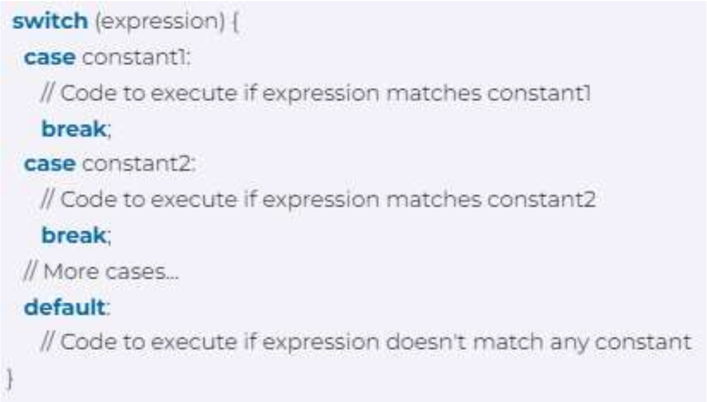
Input :

0

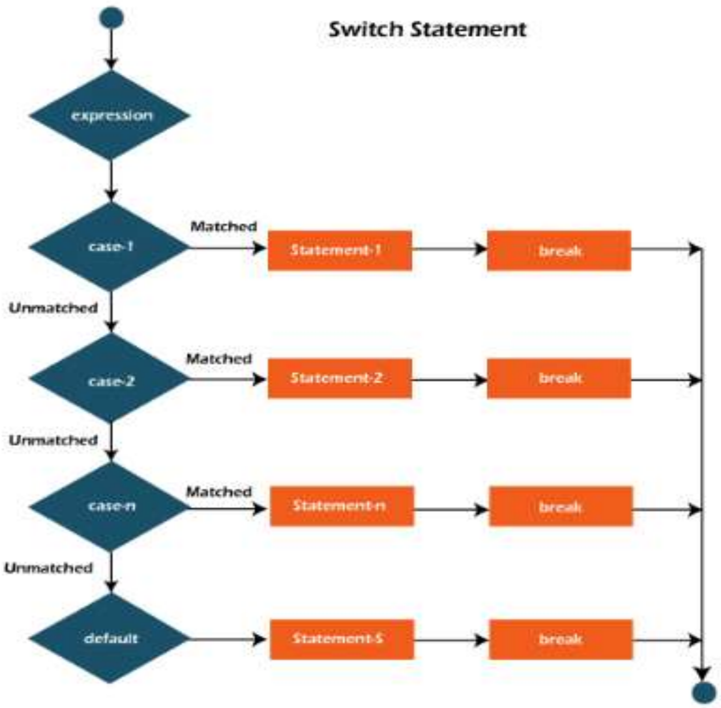
Output :

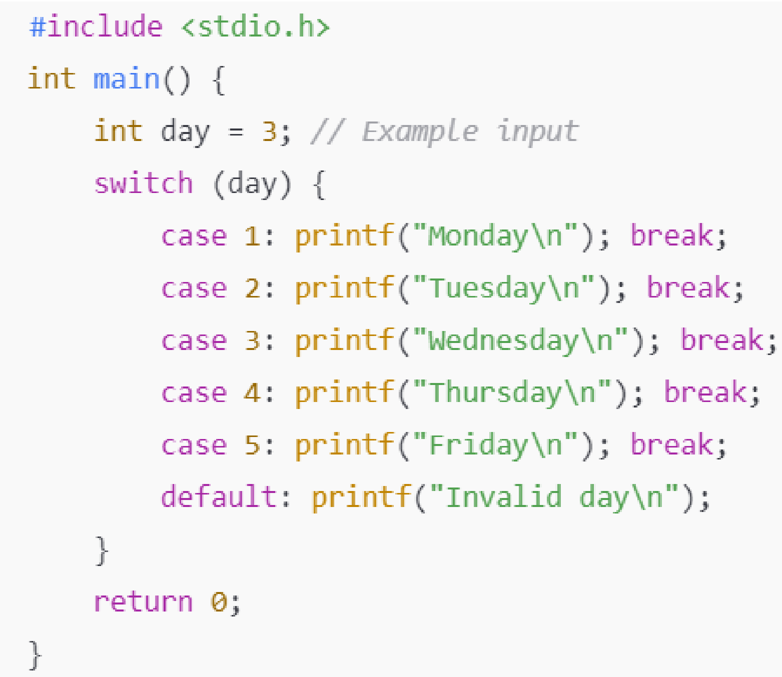
Number is Zero.

switch Statement



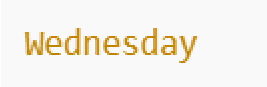
```
switch (expression) {
 case constant1:
   // Code to execute if expression matches constant1
   break;
 case constant2:
   // Code to execute if expression matches constant2
   break;
 // More cases...
 default:
   // Code to execute if expression doesn't match any constant
}
```





```
#include <stdio.h>
int main() {
    int day = 3; // Example input
    switch (day) {
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3: printf("Wednesday\n"); break;
        case 4: printf("Thursday\n"); break;
        case 5: printf("Friday\n"); break;
        default: printf("Invalid day\n");
    }
    return 0;
}
```

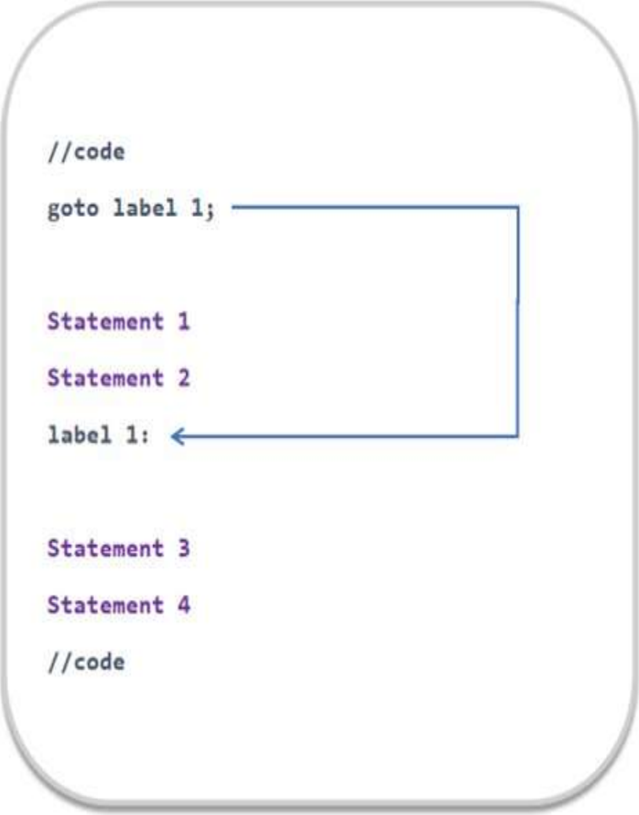
Output:



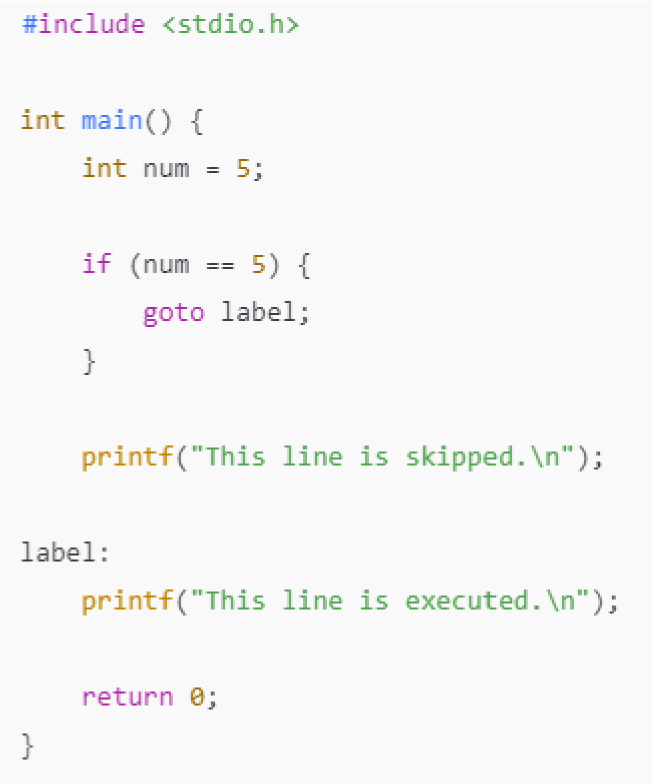
```
Wednesday
```

Unconditional branching statements (JUMP STATEMENTS)

goto Statement

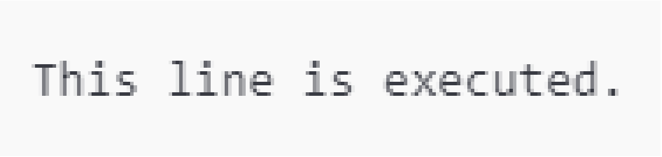


```
//code
goto label 1;

Statement 1
Statement 2
label 1:

Statement 3
Statement 4
//code
```



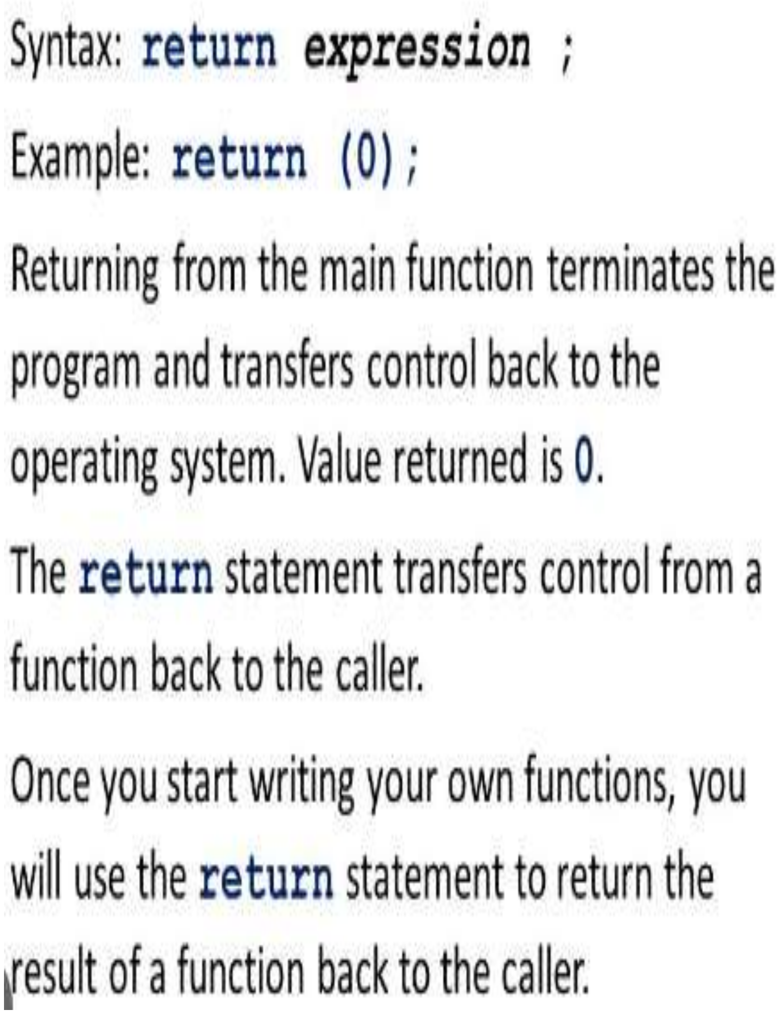
```c
#include <stdio.h>

int main() {
    int num = 5;

    if (num == 5) {
        goto label;
    }

    printf("This line is skipped.\n");

label:
    printf("This line is executed.\n");

    return 0;
}
```

Output:



```
This line is executed.
```

return Statement



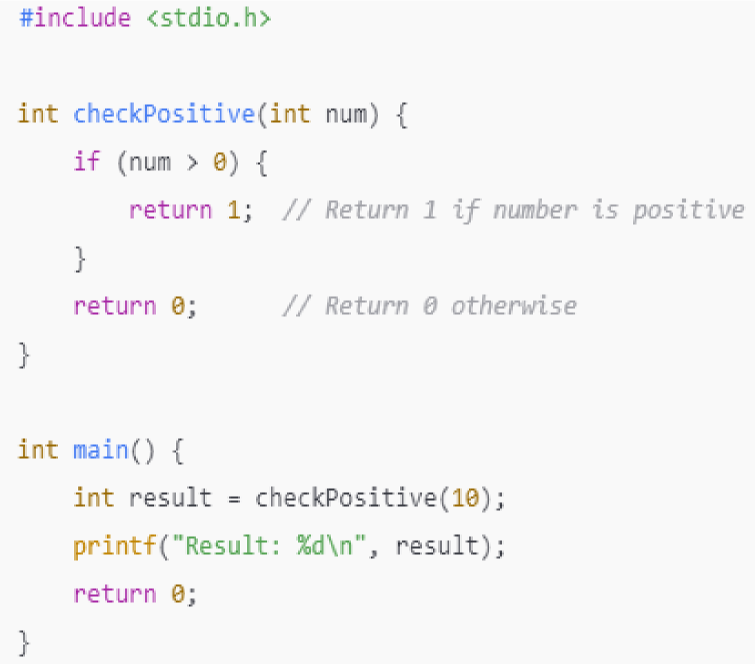
Syntax: `return expression ;`

Example: `return (0) ;`

Returning from the main function terminates the program and transfers control back to the operating system. Value returned is 0.

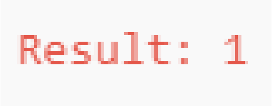
The `return` statement transfers control from a function back to the caller.

Once you start writing your own functions, you will use the `return` statement to return the result of a function back to the caller.



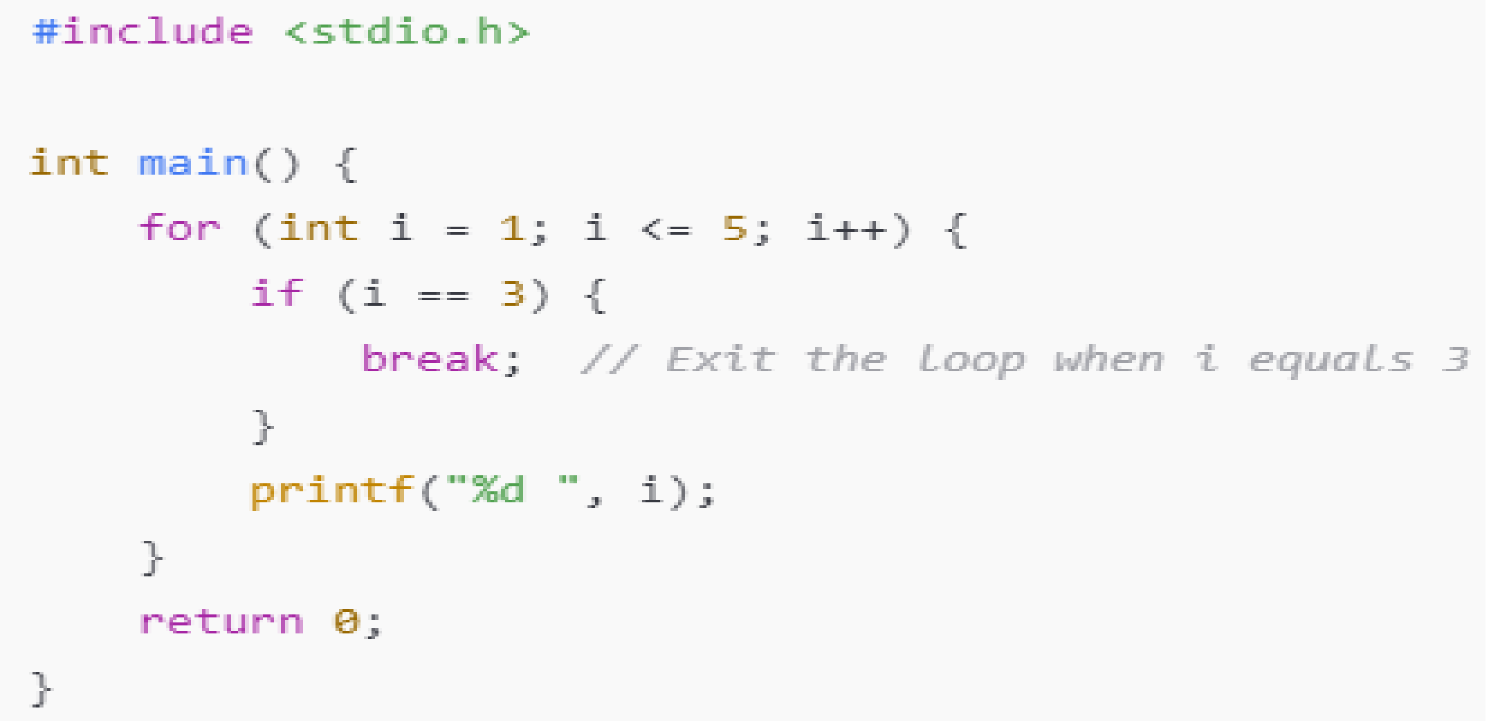
```c
#include <stdio.h>

int checkPositive(int num) {
    if (num > 0) {
        return 1;  // Return 1 if number is positive
    }
    return 0;      // Return 0 otherwise
}

int main() {
    int result = checkPositive(10);
    printf("Result: %d\n", result);
    return 0;
}
```

Output:



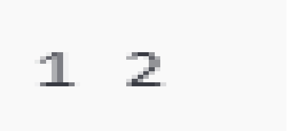
```
Result: 1
```

break Statement



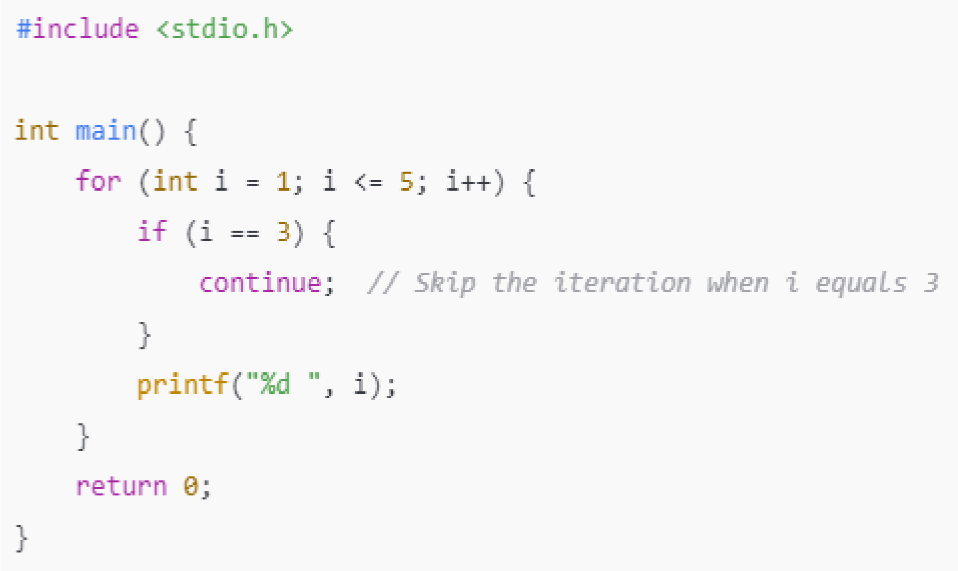
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            break;  // Exit the loop when i equals 3
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:



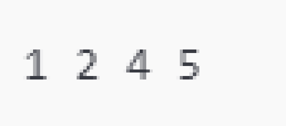
```
1 2
```

continue Statement



```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue;  // Skip the iteration when i equals 3
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:



```
1 2 4 5
```

Advantages and Disadvantages of Branching Statements

Advantages

- Better Decision Making
- Readability of the code
- Code effectiveness
- Flexibility
- Code Reusability

Disadvantages

- Code Complexity
- Readability Issues
- Potential for Logical Mistakes
- Code Maintenance

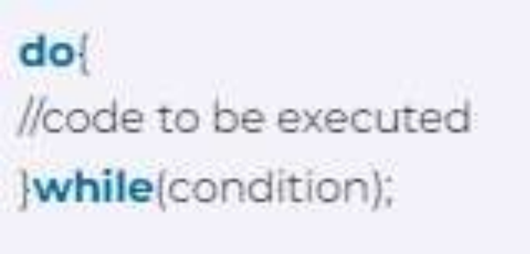
LOOPING STATEMENTS IN C

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language.

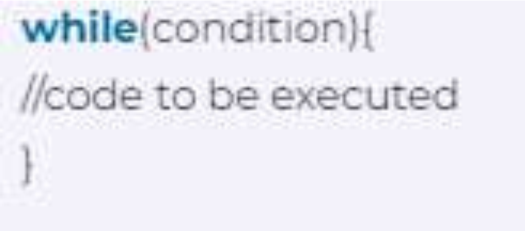
There are three types of loops in C language that is given below:

- do while
- while
- for

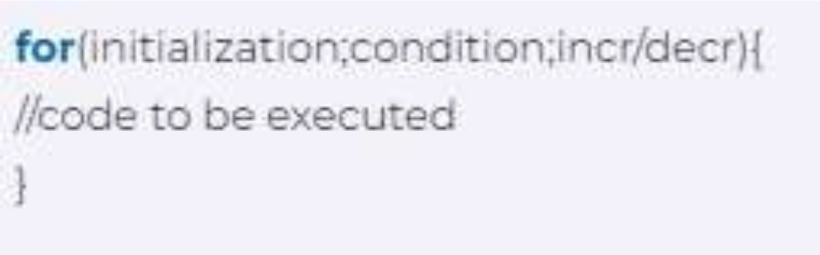
Syntax



```
do{
//code to be executed
}while(condition);
```

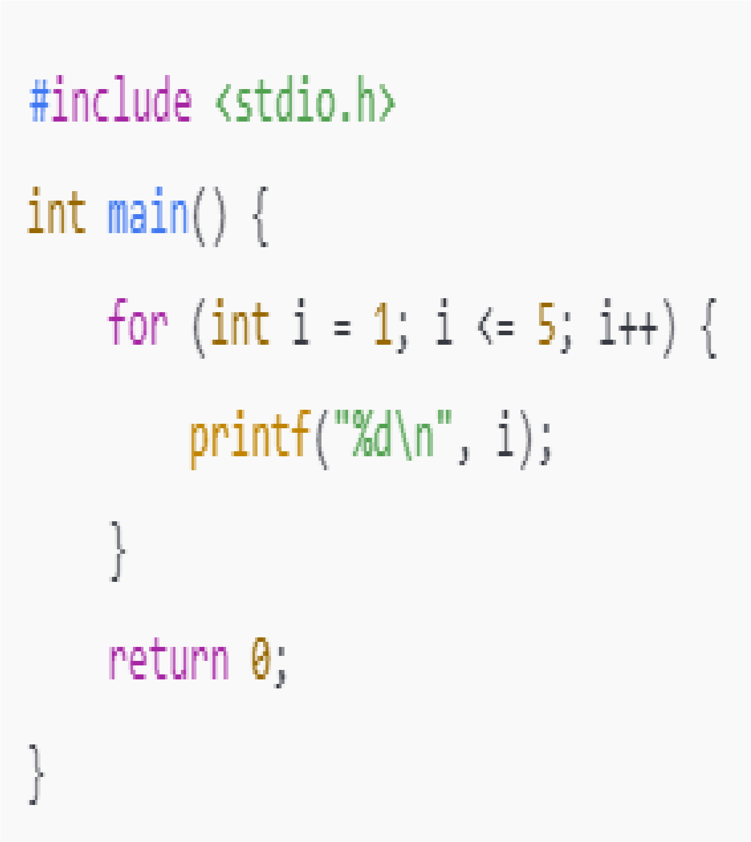


```
while(condition){
//code to be executed
}
```



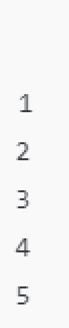
```
for(initialization;condition;incr/decr){
//code to be executed
}
```

for Loop



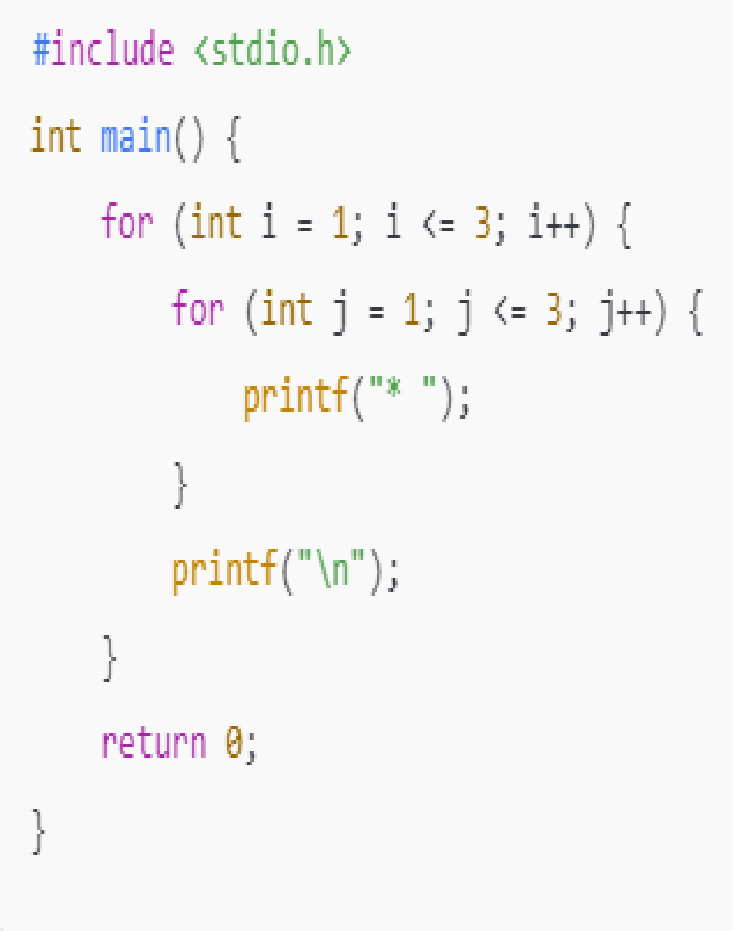
```c
#include <stdio.h>

int main() {

    for (int i = 1; i <= 5; i++) {

        printf("%d\n", i);

    }

    return 0;

}
```

Output:



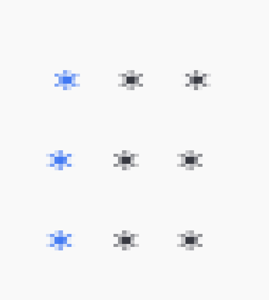
```
1
2
3
4
5
```

Nested for Loop



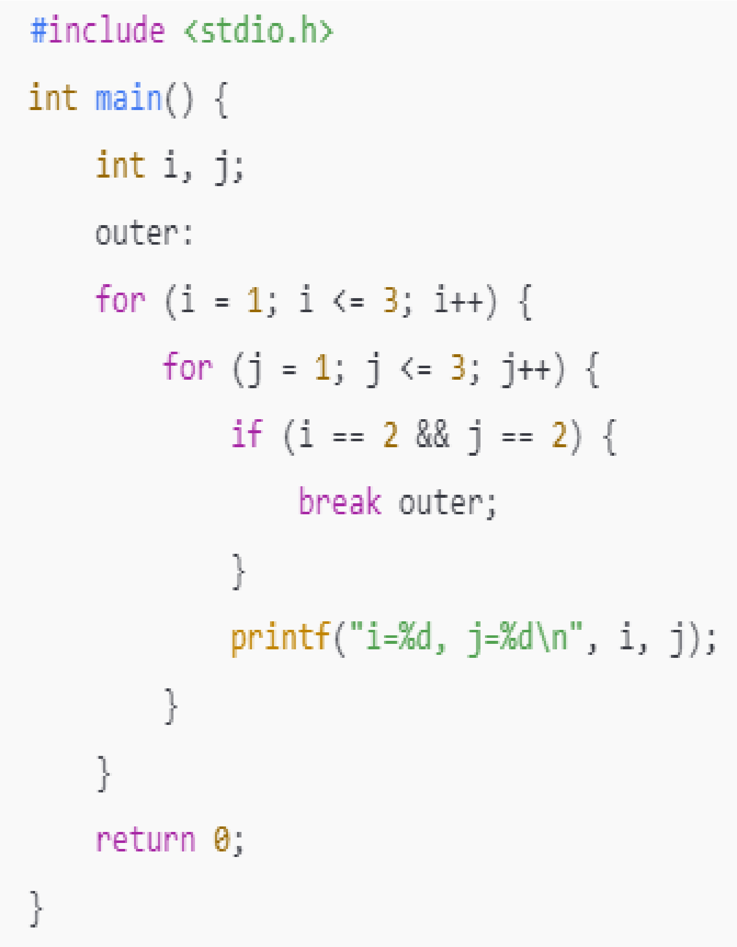
```c
#include <stdio.h>

int main() {

    for (int i = 1; i <= 3; i++) {

        for (int j = 1; j <= 3; j++) {

            printf("* ");

        }

        printf("\n");

    }

    return 0;

}
```

Output:



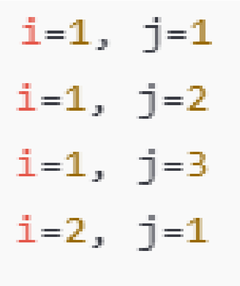
```
* * *
* * *
* * *
```

Labelled for Loop



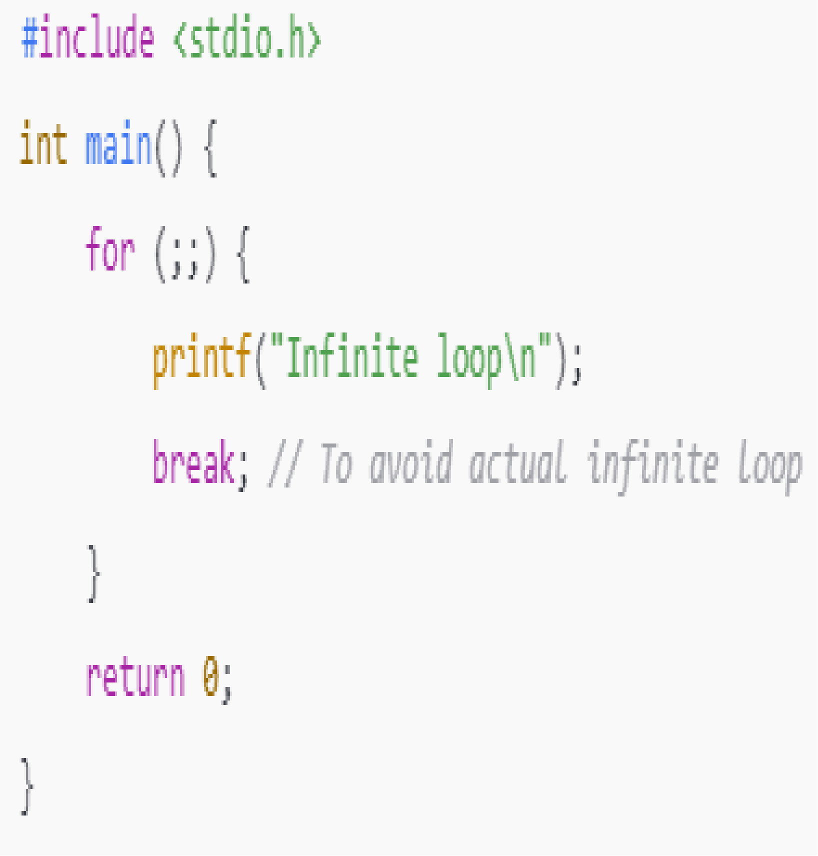
```
#include <stdio.h>
int main() {
    int i, j;
    outer:
    for (i = 1; i <= 3; i++) {
        for (j = 1; j <= 3; j++) {
            if (i == 2 && j == 2) {
                break outer;
            }
            printf("i=%d, j=%d\n", i, j);
        }
    }
    return 0;
}
```

Output:



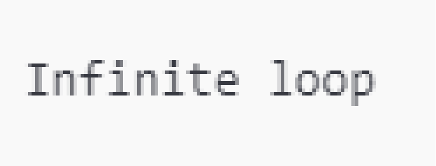
```
i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
```

Infinite for Loop



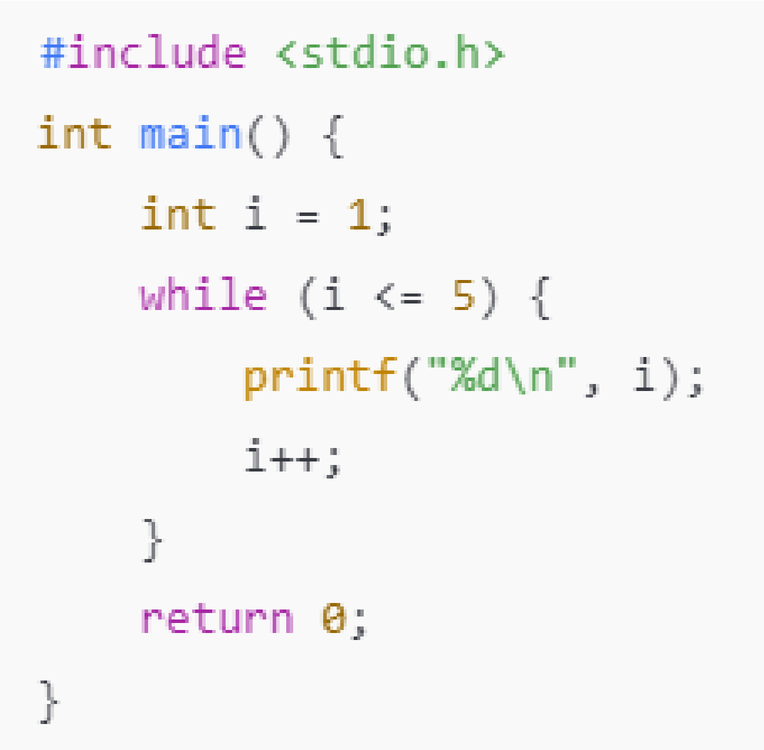
```c
#include <stdio.h>

int main() {

    for (;;) {

        printf("Infinite loop\n");

        break; // To avoid actual infinite loop

    }

    return 0;

}
```

Output:



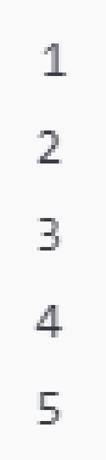
```
Infinite loop
```

while Loop



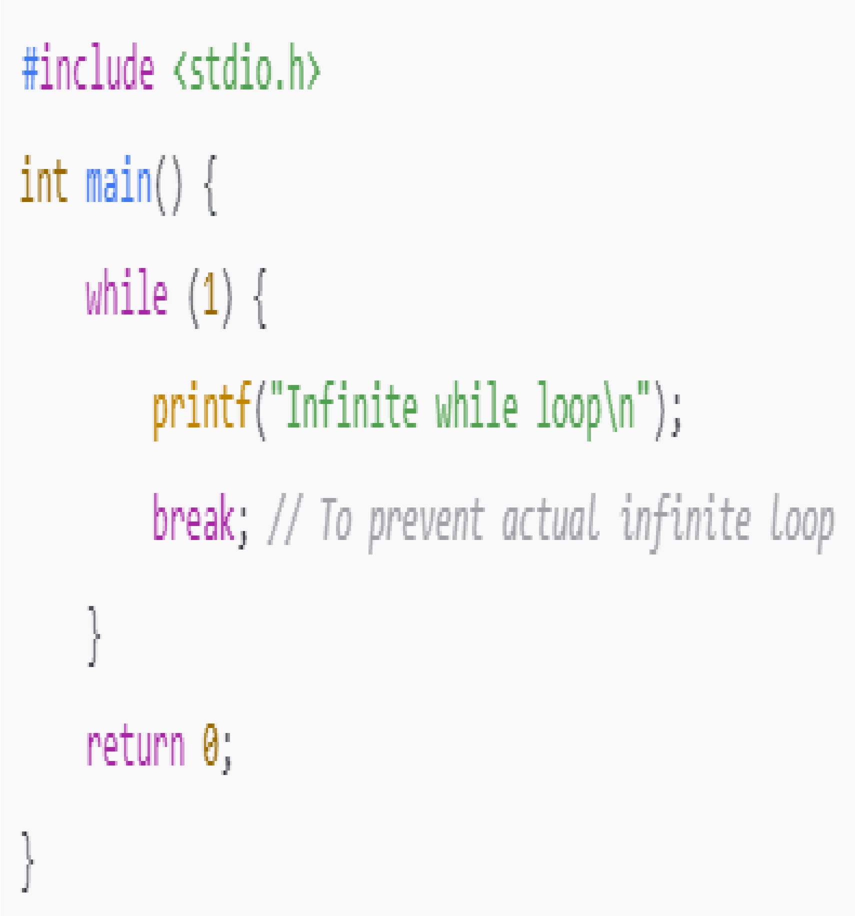
```
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Output:



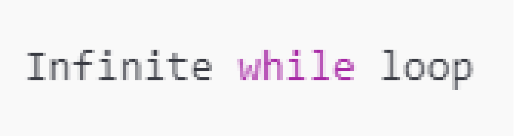
```
1
2
3
4
5
```

Infinite while Loop



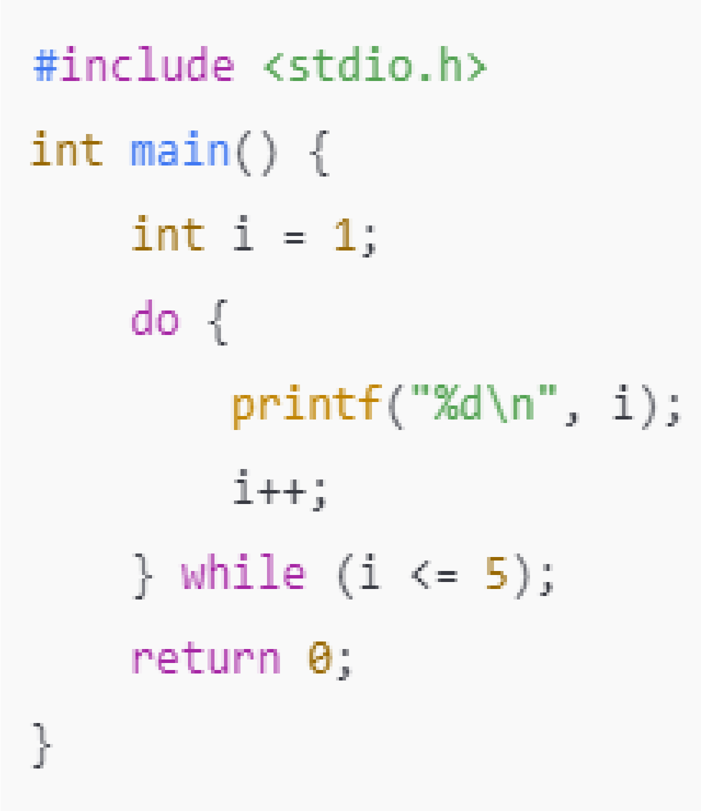
```
#include <stdio.h>

int main() {

    while (1) {

        printf("Infinite while loop\n");

        break; // To prevent actual infinite loop

    }

    return 0;

}
```

Output:



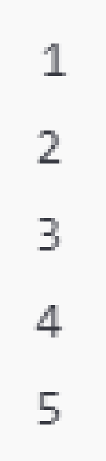
```
Infinite while loop
```

do while Loop



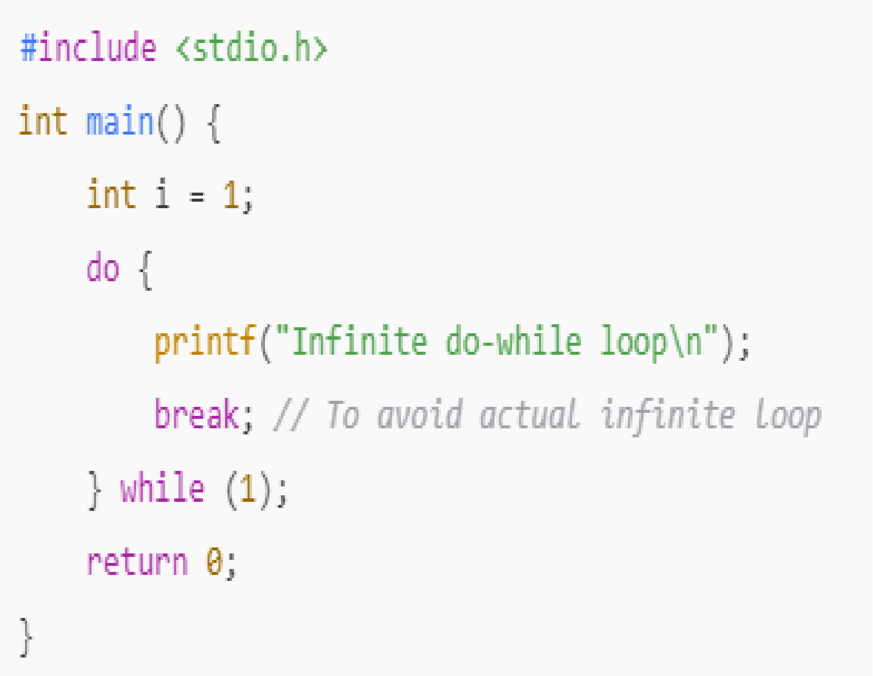
```c
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

Output:



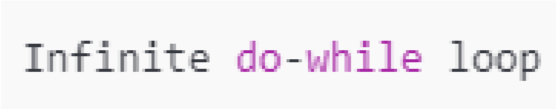
```
1
2
3
4
5
```

Infinite do while Loop



```c
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("Infinite do-while loop\n");
        break; // To avoid actual infinite loop
    } while (1);
    return 0;
}
```

Output:



```
Infinite do-while loop
```