SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

For Loop

By

Dr. Vaibhav P. Vasani

Assistant Professor

Department of Computer Engineering

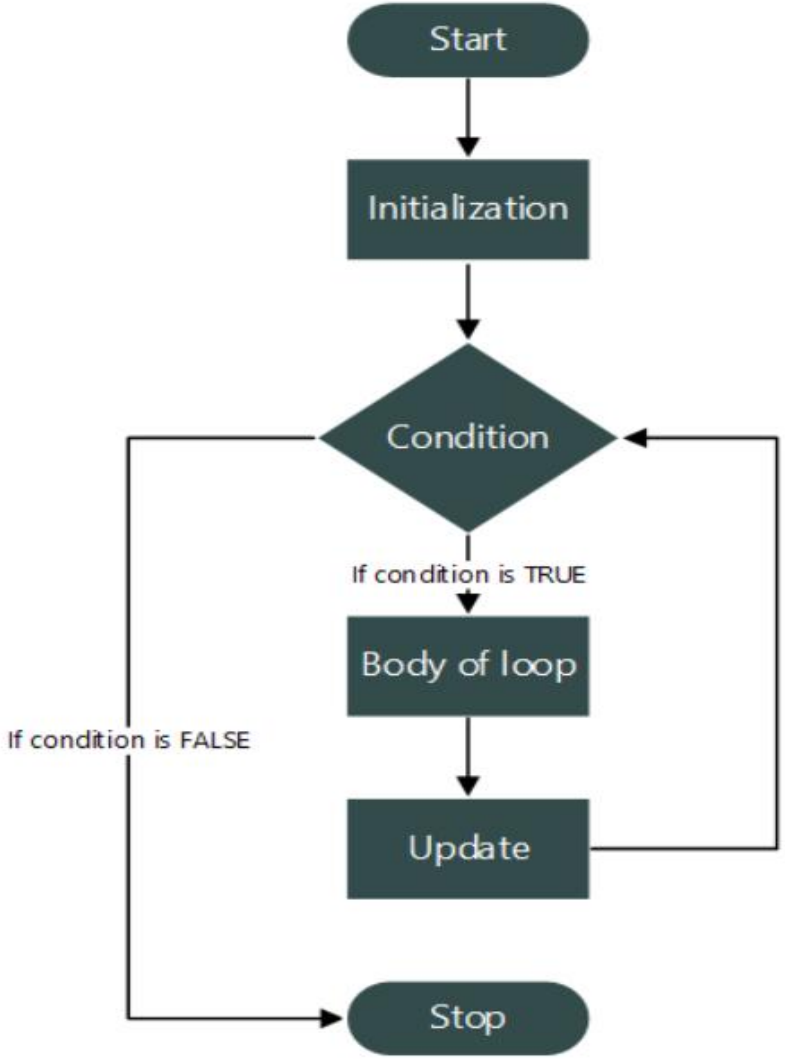
K. J. Somaiya College of Engineering

Somaiya Vidyavihar University

Somaiya TRUST

for loop Syntax

```
for (initialization; condition test; increment or
decrement)
{
//statements to be executed repeatedly
}
```



for loop Flow chart

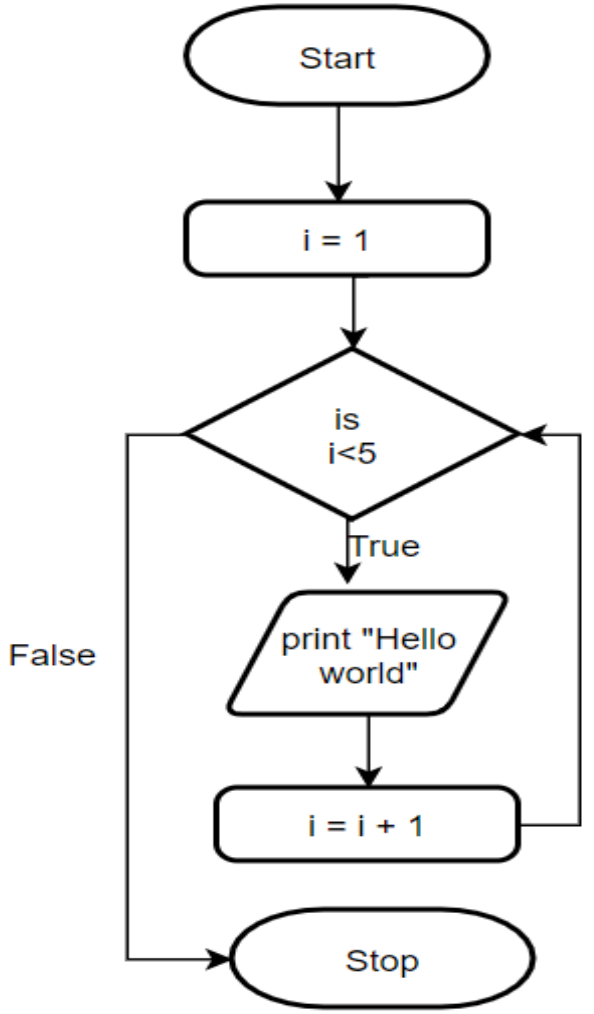
Program: for loop

Write a program to print "Hello World" 4 times using

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

Algorithm:

Step 1: Start
Step 2: i = 1
Step 3: if(i > 5)
THEN GOTO Step 7
ENDIF
Step 4: print "Hello World"
Step 5: i = i +1
Step 6: Go to Step 3
Step 7: Stop



Program: for loop

```
int main()
{
    int i;
    for(i=1;i<5;i++)
    {
    cout<<"Hello, World!\n";
    }
    return 0;
}
```

Output

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Algorithm & Flowchart to print multiplication Table of a number

Algorithm

Step-1 Start

Step-2 Input Value of NUM

Step-3 I = 1

Step-4 IF (I >10) THEN
GO TO Step 9
ENDIF

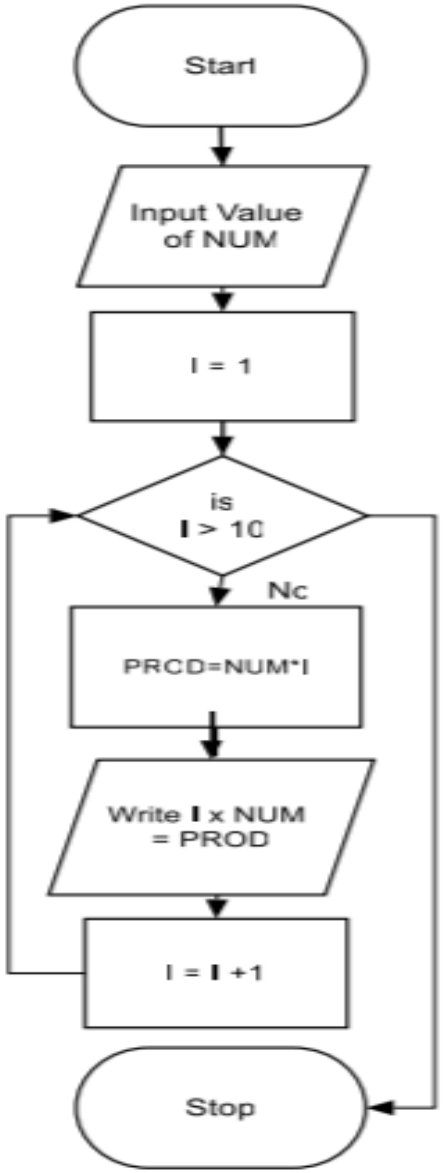
Step-5 PROD = NUM * I

Step-6 WRITE I "x" NUM "=" PROD

Step-7 I = I + 1

Step-8 Go to step-4

Step-9 Stop



Various forms of for loop in C

- 1) Here instead of num++, I'm using num=num+1 which is same as num++.
- `for (num=10; num<20; num=num+1)`

- 2) Initialization part can be skipped from loop as shown below, the counter variable is declared before the loop.
- `int num=10;`
- `for (;num<20;num++)`

Various forms of for loop in C

3) Like initialization, you can also skip the increment part as we did below. In this case semicolon (;) is must after condition logic. In this case the increment or decrement part is done inside the loop.

```
for (num=10; num<20; )
{
    //Statements
    num++;
}
```

4) This is also possible. The counter variable is initialized before the loop and incremented inside the loop.

```
int num=10;
for (;num<20;)
{
    //Statements
    num++;
}
```

Various forms of for loop in C

- 5) As mentioned above, the counter variable can be decremented as well. In the below example the variable gets decremented each time the loop runs until the condition num>10 returns false.
- `for(num=20; num>10; num--)`

Example of for loop with multiple test conditions

```
int main()
{
    int i,j;
    for (i=1,j=1 ; i<3 || j<5; i++,j++)
    {
      cout<<i<<j;
    }
    return 0;
}
```

Programs on for loop

1. Write a program to print numbers from 1 to 10
2. Write a program to calculate sum of first n natural numbers
3. Write a program in C to read 10 numbers from keyboard and find their sum an average
4. WAP to print N Fibonacci number up to N using for loop.
5. Write a program in C to display the cube of the number upto a given integer.
6. Write a program in C to display the multiplication table of a given integer

```
int main()
 {
    int a=0, b=1, num, c, sum=0;

    cout<<“Enter the number of terms:”;
    cin>>num;

    cout<<"The fibonacci series is: \t”;

    for(i=1; i<=num; i++) /* prints series for n number of terms */
    {
       cout<<a;
       sum = sum + a;
       c = a + b;
       a = b;
       b = c;
    }
}
```

output for num =9?

0 1 1 2 3 5 8 13 21

Nested Loops

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

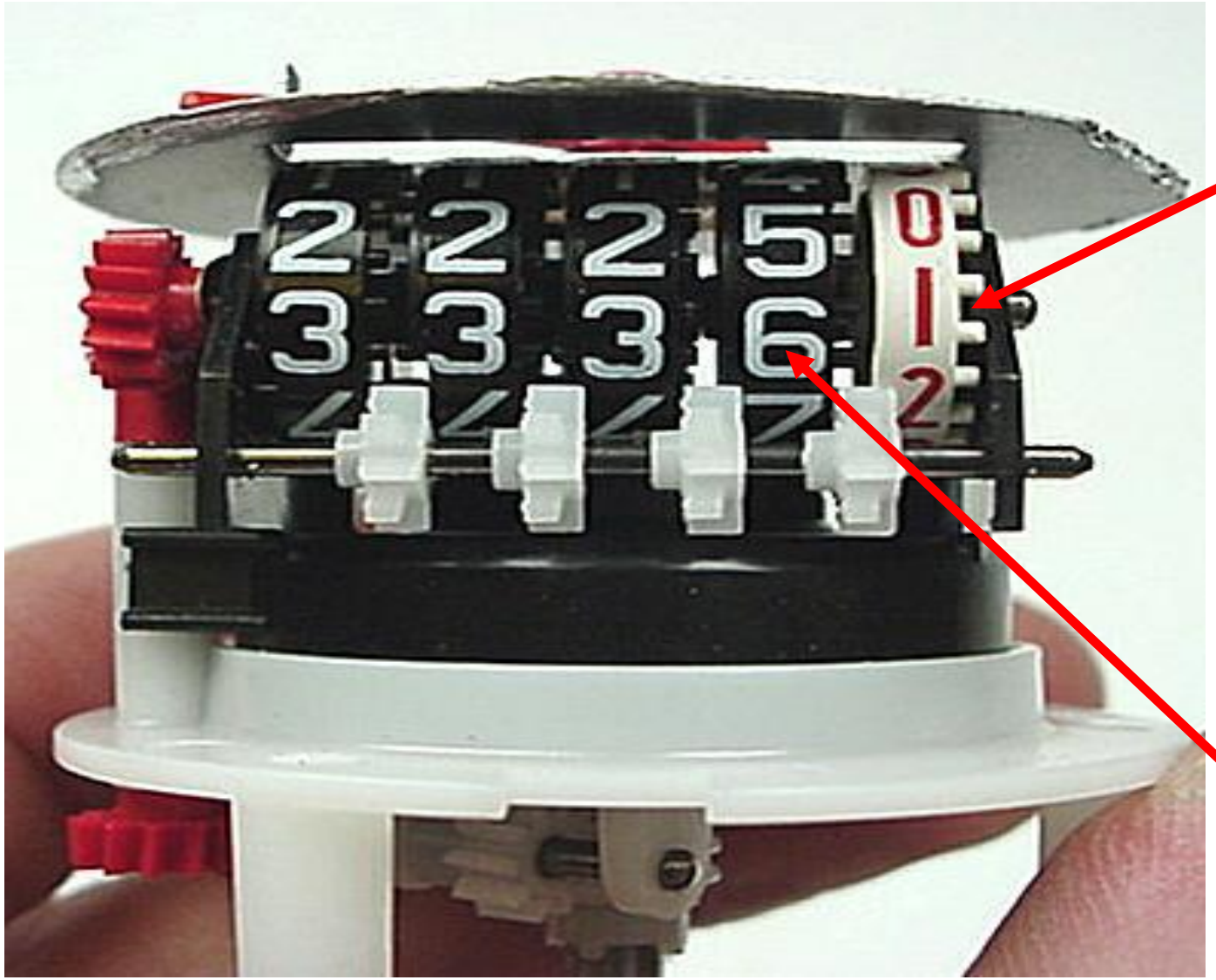
```
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 <= 20)
    {
        cout<<"Here";
        count2++;
    }
    count1++;
}
```

10 * 20 = 200

Analogy for Nested Loops

http://www.brandondufau.com/archives/odometer%201.jpg

Analogy for Nested Loops



http://static.howstuffworks.com/gif/odometer2.jpg

Nested For Loop in C

- **Its Show time**

Nested For Loop

write logic for sorting elements in the array

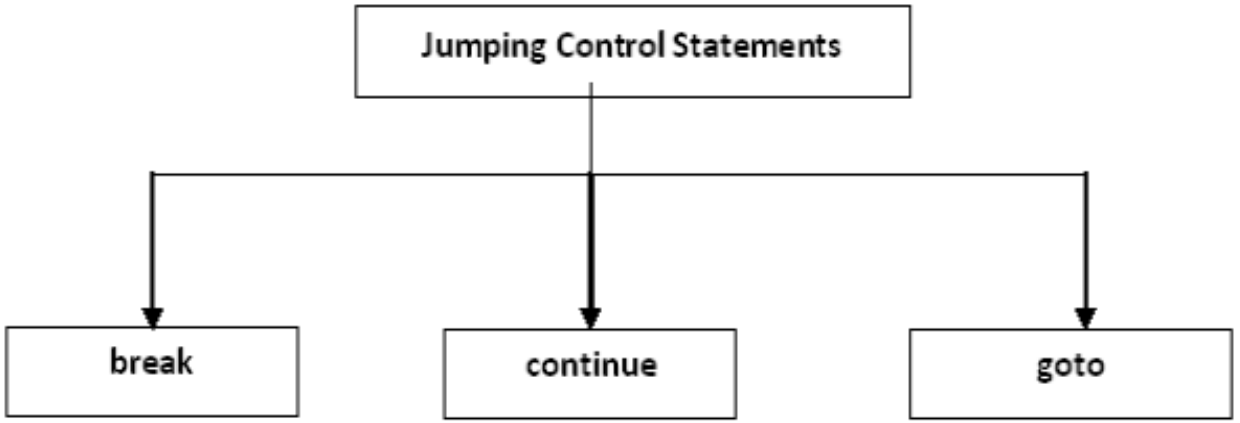
```
for(i=0;i<n-1;i++)
    {
       for(j=0;j<n-1-i;j++)
       {
          if(arr[j]>arr[j+1])
          {
             temp=arr[j];
             arr[j]=arr[j+1];
             arr[j+1]=temp;
          }
       }
    }
```

What's the output for following code

```
for (i=0;i<2;i++)
{
for(j=0;j<4;j++)
{
cout<<i<<j;
}
}
```

Jumping control-flow statements.

Jumping control-flow statements are the control-flow statements that transfer the control to the specified location or out of the loop or to the beginning of the loop. There are 3 jumping control statements:



Jumping control-flow statements.

The "break" statement is used with in the looping control statements, switch statement and nested loops. When it is used with the for, while or do-while statements, the control comes out of the corresponding loop and continues with the next statement.

```
Any loop
{
    statement_1;
    statement_2;
        :
    break;
        :
}
next_statement
```

```
#include <iostream>

int main() {
   for (int i = 1; i <= 10; ++i) {
      if (i == 6) {
         break; // Exit the loop
      }
      std::cout << i << " ";
   }
   return 0;
}
```

Jumping control-flow statements.

A continue statement is used within loops to end the execution of the current iteration and proceed to the next iteration. It provides a way of skipping the remaining statements in that iteration after the continue statement.

```
Any loop
{
    statement_1;
    statement_2;
        :
    continue;
        :
}
next_statement
```

```
#include <iostream>
int main() {
   int n;
   int sum = 0;

   // This loop will run 10 times
   for (int i = 1; i <= 10; ++i) {
      std::cout << "Enter a number: ";
      std::cin >> n;

      // If the number is negative, skip the addition
      if (n < 0) {
         std::cout << "Negative number skipped." << std::endl;
         continue; // Go to the next loop iteration
      } else {
         // Add the non-negative number to the sum
         sum = sum + n; // Or use the shorthand: sum += n;
      }

      // The original code prints the sum on every iteration
      std::cout << "Running total: " << sum << std::endl;
   }
   return 0;
}
```

Jumping control-flow statements.

```
#include <iostream>
int main() {
   for (int i = 0; i < 10; ++i) {
      // If i is 5, skip the rest of this iteration
      // and go to the next one (i=6).
      if (i == 5) {
         continue;
      }

      // Print the current value of i.
      // I've added a space for readability.
      std::cout << i << " ";

      // If i is 8, print it and then
      // terminate the loop immediately.
      if (i == 8) {
         break;
      }
   }
   std::cout << std::endl; // For a clean newline at the end.
   return 0;
}
```

Jumping control-flow statements.

```
#include <iostream>

int main() {
   for (int i = 0; i < 10; ++i) {
      // If i is 5, skip the rest of this iteration
      // and jump to the next one (where i becomes 6).
      if (i == 5) {
         continue;
      }

      // Print the current value of i.
      // I've added a space for better readability.
      std::cout << i << " ";

      // If i is 8, print it first, and then
      // terminate the loop immediately.
      if (i == 8) {
         break;
      }
   }
   std::cout << std::endl; // For a clean newline at the end.
   return 0;
}
```

Jumping control-flow statements.

Difference Between break and continue:

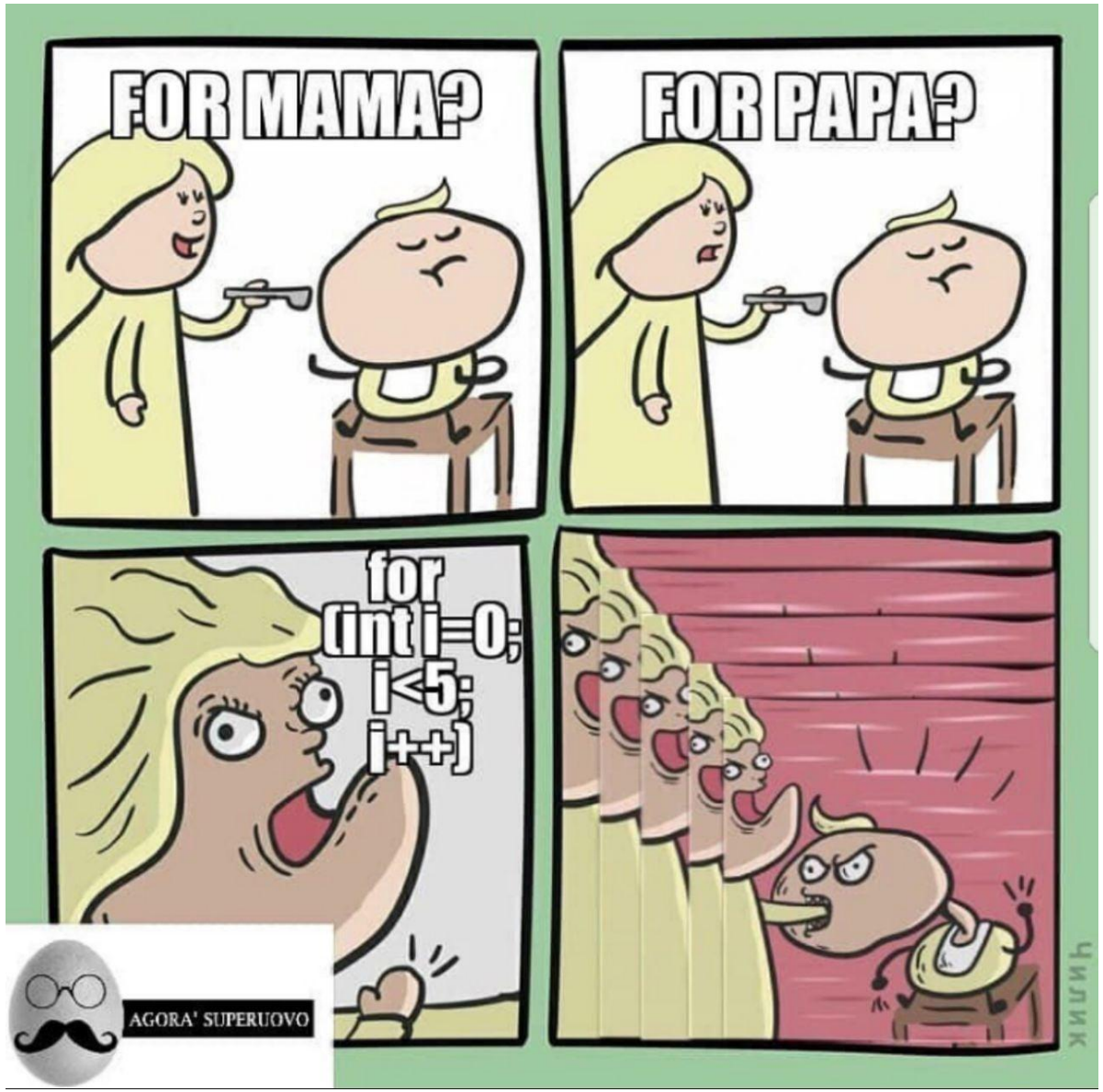
break	continue
A `break` can appear in both `switch` and loop (`for`, `while`, `do`) statements.	A `continue` can appear only in loop (`for`, `while`, `do`) statements.
A `break` causes the `switch` or loop statements to terminate the moment it is executed. Loop or `switch` ends abruptly when break is encountered.	A `continue` doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if `continue` is encountered. The `continue` statement is used to skip statements in the loop that appear after the `continue`.
When a `break` statement is encountered, it terminates the block and gets the control out of the `switch` or loop.	When a `continue` statement is encountered, it gets the control to the next iteration of the loop.
A `break` causes the innermost enclosing loop or `switch` to be exited immediately.	A `continue` inside a loop nested within a `switch` causes the next loop iteration.

Jumping control-flow statements.

The goto statement transfers the control to the specified location unconditionally. There are certain situations where goto statement makes the program simpler. For example, if a deeply nested loop is to be exited earlier, goto may be used for breaking more than one loop at a time. In this case, a break statement will not serve the purpose because it only exits a single loop.

```
label:
        {
        statement_1;
        statement_2;
                :
        }
        :
    goto label;
```

- In this syntax, goto is the keyword and label is any valid identifier and should be ended with a colon (:).
- The identifier following goto is a statement label and need not be declared. The name of the statement or label can also be used as a variable name in the same program if it is declared appropriately.


FOR MAMA?
FOR PAPA?
for (int i=0; i<5; i++)
AGORA' SUPERUOVO
Чилик

Thank you