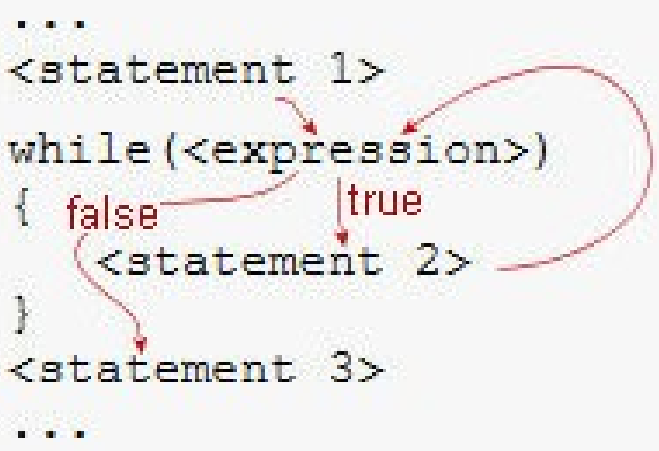
LOOP

while

```
...
<statement 1>
while(<expression>)
{
   <statement 2>
}
<statement 3>
...
```



while

- <expression> is checked to decide whether the <statement2> block will be executed or not.
- After the execution of <statement2> block, the process always go back to <expression> evaluation.
- Once <expression> returns False value, the process jump to <statement3>

do-while

```
...
<statement 1>
do
{
   <statement 2>
} while(<expression>);
<statement 3>
...
```

```
...
<statement 1>
do
{
   <statement 2>

} while(<expression>);

<statement 3>
...
```

do-while

- <statement2> block is first executed, then the <expression> evaluation.
- <expression> is checked to decide whether the <statement2> block will be executed again or not.
- Note: ‘do-while’ form will always execute <statement2> block at least once, on the other hand, ‘while’ form might not even execute <statement2> block

for

```
...
<statement 1>
for(<expression1>; <expression2>; <expression3>)
{
   <statement 2>
}
<statement 3>
...
```

```
<statement 1>

for(<expression1>; <expression2>; <expression3>)
{
   <statement 2>
}
<statement 3>
```

for

- <expression1> is loop control-variable initialization, and executed only once.
- <expression2> is loop control-variable evaluation
- <expression3> is loop control-variable operation. This expression is executed after <statement2>, just before <expression2> evaluation

Average, in ‘while’ form

```
#include <stdio.h>
main()
{
  int ntot, n=1;
  float num, avg, sum=0;
  printf("\nSpecify how many numbers you will feed: ");
  scanf("%d", &ntot);
  while(n <= ntot) {
    printf("Give Number %d: ",n);
    scanf("%f", &num);
    sum += num;
    n++;
  }
  avg = sum/ntot;
  printf("\nThe average is %f\n", avg);
}
```

Average, in ‘do-while’ form

```
#include <stdio.h>
main()
{
  int ntot, n=1;
  float num, avg, sum=0;
  printf("\nSpecify how many numbers you will feed: ");
  scanf("%d", &ntot);
  do {
    printf("Give Number %d: ",n);
    scanf("%f", &num);
    sum += num;
    n++;
  } while(n <= ntot);
  avg = sum/ntot;
  printf("\nThe average is %f\n", avg);
}
```

Average, in ‘for’ form

```
#include <stdio.h>
main()
{
  int ntot, n;
  float num, avg, sum=0;
  printf("\nSpecify how many numbers you will feed: ");
  scanf("%d", &ntot);
  for(n = 1; <= ntot; n++) {
    printf("Give Number %d: ",n);
    scanf("%f", &num);
    sum += num;
  }
  avg = sum/ntot;
  printf("\nThe average is %f\n", avg);
}
```

Example: Palindrome Check

```
#include <stdio.h>
main()
{
  int number, digit, reverse=0, store_num;
  printf("Give a Number:");
  scanf("%d", &number); fflush(stdin);
  store_num = number;
  do {
    digit = number % 10;
    reverse = (reverse * 10) + digit;
    number /= 10;
  } while(number != 0);
  number = store_num;
  if(number == reverse)
    printf("The number is a palindrome.\n");
  else
    printf("The number is not a palindrome.\n");
}
```

Comma Operator

This operator is used in a for statement to separate multiple statements in the expressions within for(<expression1>; <expression2>; <expression3>)

```
for(i=1,j=2; i<5,j<10; ++i,j++)
{
        ...
}
```

Note that the comma operator is only relevant in case of for loop construct, not for while and do-while.

break & continue statements

- The break statement is used to jump out of a loop. You would have also noticed the usage of break earlier in the switch construct.
- The continue statement is used to bypass/skip the remaining statements in a loop and jump to the next pass.

goto

- The ‘goto’ statement can be used to unconditionally alter the program execution sequence.
- As a result of the ‘goto’ statement, the program execution jumps to a new statement which is identified by a ‘label’.

goto

- Example

```
#include <stdio.h>
main()
{
        int a, b, sum;
        ...
        ...
        sum += (a + b);
        goto printf_statement;
        ...
        ...
        ...
        printf_statement : printf("sum is %d\n", sum);
        ...
        ...
}
```

goto

- Although the usage of goto may at times be convenient to programmers, it is desirable that usage of such statement is avoided as far as possible.
- An unrestricted usage of goto statement, makes program debugging very difficult and may at times introduce logical errors.

END
Q&A