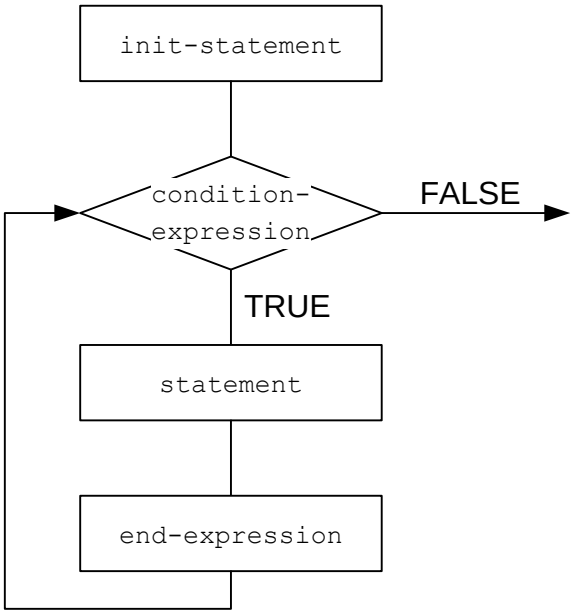
FOR LOOP

By far, the most utilized looping statement in C++ is the **for** statement. The for statement (also called a **for loop**) is ideal when we know exactly how many times we need to iterate, because it lets us easily define, initialize, and change the value of loop variables after each iteration.

The for statement looks pretty simple in abstract:

```
for (init-statement; condition-expression; end-expression)
   statement;
```



The variables defined inside a for loop have a special kind of scope called loop scope. Variables with loop scope exist only within the loop, and are not accessible outside of it.

Evaluation of for statements

A for statement is evaluated in 3 parts:

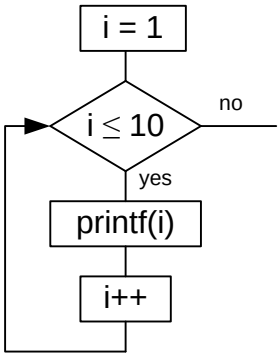
1) The **init-statement** is evaluated. Typically, the init-statement consists of variable definitions and initialization. This statement is only evaluated once, when the loop is first executed.

2) The **condition-expression** is evaluated. If this evaluates to false, the loop terminates immediately. If this evaluates to true, the statement is executed.

3) After the statement is executed, the **end-expression** is evaluated. Typically, this expression is used to increment or decrement the variables declared in the init-statement. After the end-expression has been evaluated, the loop returns to step 2.

Let's take a look at a sample for loop and discuss how it works:

```c
#include <stdio.h>

int i;

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



Initialization $i = 1$.
Iteration while condition $i \leq 10$ is ***true***, execute the body of the for loop:

- `printf("%d ",i);`
- `i++;`

The loop stops when $i = 11$. For this value of i the condition $i \leq 10$ becomes ***false***.

E-OLYMP **5325. Stand in order** Print all integers from 1 to n inclusively.
► Run variable i in the for loop from 1 to n and print i.

E-OLYMP **8931. All OK** Count from 1 to n and for each number print the message **OK.**
► Run for loop from 1 to n.

E-OLYMP **8932. Marathon 2** Print numbers from a to b in increasing order.
► Run for loop from a to b in increasing order.

E-OLYMP **8933. All OK** Count from n to 0 and for each number print the message **sek.**
► Run for loop from n to 0 in decreasing order.

E-OLYMP **8934. Marathon 3** Print numbers from a to b in decreasing order.
► Run for loop from a to b in decreasing order.

Find the sum of numbers from 1 to n:

```c
#include <stdio.h>

int s, i, n;

int main(void)
{
  scanf("%d",&n);
  s = 0;
  for (i = 1; i <= n; i++)
    s = s + i;
  printf("%d\n",s);
  return 0;
```

```
}
```

Let's find the value of *s* after the `for` loop for the next program. Here variable *i* starts from 6, and runs with step 3, so *i* = 6, 9, 12, 15, 18, 21. If *i* is odd, it is added to the sum *s*. Initial value of *s* is not zero, it is 3.

At each iteration first we process the body of the `for` loop, and then *i* is increased by 3.

```
#include <stdio.h>

int s, i;

int main(void)
{
  s = 3;
  for (i = 6; i < 20; i = i + 3)
    if (i % 2 == 1) s = s + i;
  printf("%d\n",s);
  return 0;
}
```

	i	s	
init	6	3	
1 iteration	9	3	6 is even, nothing is added to s
2 iteration	12	12	s = s + i = 3 + 9 = 12
3 iteration	15	12	12 is even, nothing is added to s
4 iteration	18	27	s = s + i = 12 + 15 = 27
5 iteration	21	27	21 < 20 is false, stop for loop

Omitted expressions

It is possible to write for loops that omit any or all of the expressions. For example:

```
#include <stdio.h>

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

Rather than having the for loop do the initialization and incrementing, we've done it manually. We have done so purely for academic purposes in this example, but there

are cases where not declaring a loop variable (because you already have one) or not incrementing it (because you're incrementing it some other way) are desired.

Although you do not see it very often, it is worth noting that the following example produces an infinite loop:

```
for (;;)
  statement;
```

Sequence of numbers processing – minimum and sum of *n* numbers

Given sequence of *n* integers. Find their minimum. First line contains number *n*. Next line contains *n* integers. Print the minimum among the elements of the sequence.

Sample input	**Sample output**
4 5 8 -4 6	-4

Let's initialize the minimum value *mn* (result of the program) with some big value, called infitity (∞). Let is be *mn* = 2000000000 ($2 * 10^9$, this value is about the upper value for **int** data type). Now for each value *val*, if it is less then *mn*, assign *val* to *mn*.

```
#include <stdio.h>

int i, n, val, mn;

int main(void)
{
  scanf("%d",&n);
  mn = 2000000000; // initialize mn with max value
  for(i = 1; i <= n; i++)
  {
    scanf("%d",&val);
    if (val < mn) mn = val;
  }
  printf("%d\n",mn);
  return 0;
}
```

Given sequence of *n* real numbers. Find the sum of all its elements. First line contains number *n*. Next line contains *n* real numbers. Print the sum of all sequence elements.

Sample input	**Sample output**
5 1.2 1.3 5.7 1.8 12.4	22.4

```
#include <stdio.h>

int i, n;
double s, val;
```

```
int main(void)
{
  scanf("%d",&n);

  s = 0;
  for(i = 0; i < n; i++)
  {
    scanf("%lf",&val);
    s = s + val;
  }

  printf("%.1lf\n",s);
  return 0;
}
```

Multiple test cases

Find the sum of two numbers. First line contains the number of test cases *t*. Each test consists of two integers *a* and *b*. For each tests case print in a separate line the sum of two numbers *a* and *b*.

Sample input	Sample output
3 2 3 17 -18 5 6	5 -1 11

```
#include <stdio.h>

int i, k, a, b;

int main(void)
{
  scanf("%d",&k);
  for(i = 0; i < k; i++)
  {
    scanf("%d %d",&a,&b);
    printf("%d\n",a+b);
  }
  return 0;
}
```

Multiple declarations

Although for loops typically iterate over only one variable, sometimes for loops need to work with multiple variables. When this happens, the programmer can make use of the comma operator in order to assign (in the init-statement) or change (in the end-statement) the value of multiple variables:

```
#include <stdio.h>

int i, j;

int main(void)
{
```

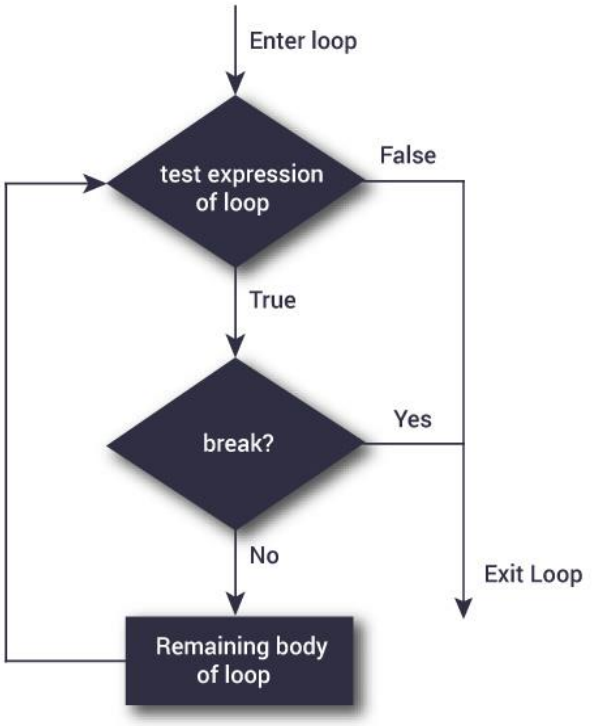
```
  for (i = 0, j = 9; i < 10; i++, j--)
    printf("%d %d\n",i,j);
  return 0;
}
```

This loop assigns values to two previously declared variables: *i* to 0, and *j* to 9. It iterates *i* over the range 0 to 9, and each iteration *i* is incremented and *j* is decremented.

Break and Continue

Break

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. The ***break*** statement terminates the loop (***for***, ***while***) immediately when it is encountered. The break statement is used with decision making statement such as ***if…else***.




```
while (test Expression)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

```
for (init, condition, update)
{
    // codes
    if (condition for break)
    {
        break;
    }
    // codes
}
```

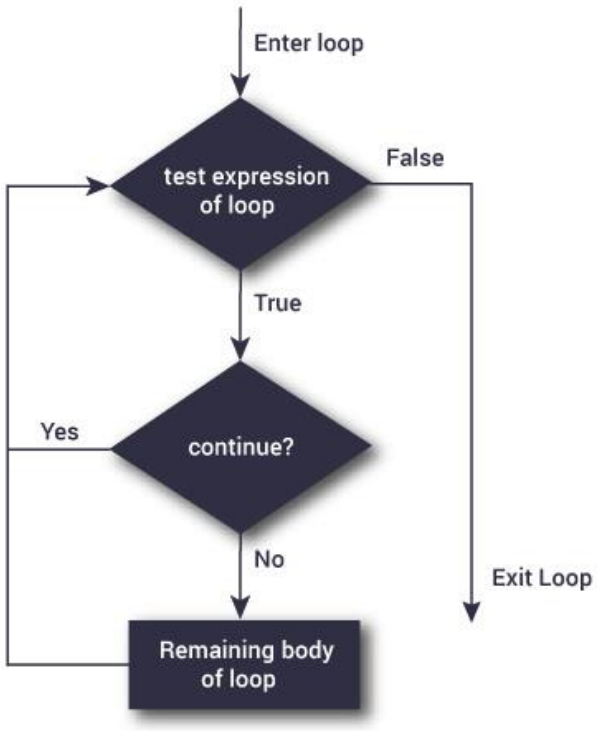
Next program calculates the sum of integers until user enters nonnegative number.

```
#include <stdio.h>

int s, x;

int main(void)
{
  s = 0;
  while(true)
  {
    scanf("%d",&x);
    // if user enters negative number, loop is terminated
    if (x < 0) break;
```

```
    s = s + x;
  }
  printf("%d\n",s);
  return 0;
}
```

Continue

The ***continue*** statement skips some statements inside the loop. The continue statement is used with decision making statement such as ***if…else***.




```
while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

```
for (init, condition, update)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

Next program calculates the sum of maximum of 5 numbers. Negative numbers are skipped from calculation.

```
#include <stdio.h>

int i, s, x;

int main(void)
{
  s = 0;
  for(i = 0; i < 5; i++)
  {
    scanf("%d",&x);
    // if user enters negative number, loop is continued
    if (x < 0) continue;
    s = s + x;
  }
  printf("%d\n",s);
  return 0;
}
```

QUIZ 1. What is the answer in the next part of the code?

```
s = 2;
for (i = 3; i <= 11; i += 2)
  s = s + 3;
printf("%d\n", s);
```

```
s = 5;
for (i = 5; i < 20; i += 5)
  s = s + 2;
printf("%d\n", s);
```

```
s = 1;
for (i = 15; i > -5; i -= 5)
  s = s * 2;
printf("%d\n", s);
```

```
s = 1;
for (i = 8; i > -2; i -= 3)
  s = s * 3;
printf("%d\n", s);
```

QUIZ 2. What is the answer in the next part of the code?

```
s = 0;
for (i = 3; i < 20; i += 3)
{
  s = s + 3;
  i = i + 2;
}
printf("%d\n", s);
```

```
s = 1;
for (i = 5; i < 25; i += 4)
{
  s = s + 2;
  i++;
}
printf("%d\n", s);
```