The Three Loops in C++

C++ has these three looping statements:

```
while
for
do
```

The `while` Loop

```
while (condition)
{
    statements
}
```

The *condition* is some kind of test
(the same as it was in the `if` statement)

The Complete Investment Program

```
#include <iostream>
using namespace std;

int main()
{
   const double RATE = 5;
   const double INITIAL_BALANCE = 10000;
   const double TARGET = 2 * INITIAL_BALANCE;

   double balance = INITIAL_BALANCE;
   int year = 0;

   while (balance < TARGET)
   {
      year++;
      double interest = balance * RATE / 100;
      balance = balance + interest;
   }

   cout << "The investment doubled after "
        << year << " years." << endl;

   return 0;
```

Program Run

Check the loop condition

balance = 10000

year = 0

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10000
year =	0
interest =	?

Program Run

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10000
year =	1
interest =	?

Program Run

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10000
year =	1
interest =	500

Program Run

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10500
year =	1
interest =	500

Program Run

The condition is true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

balance =	10500
year =	1
interest =	500

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

Check the loop condition

balance = 10500

year = 1

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10500
year =	1
interest =	?

Program Run

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10500
year =	2
interest =	?

Program Run

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	10500
year =	2
interest =	525

Program Run

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

balance =	11025
year =	2
interest =	525

Program Run

The condition is still true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Execute the statements in the loop

balance =	11025
year =	2
interest =	525

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

Check the loop condition

balance = 11025

year = 2

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

…this process goes on for 15 iterations…

…until the `balance` is finally(!) over $20,000 and the test becomes `false`.

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.82	19799.32	14
19799.32	14	989.97	20789.28	15
		while statement is over		

Program Run

After 15 iterations

balance = 20789.28

year = 15

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

balance = 20789.28

year = 15

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

The condition is no longer true

Program Run

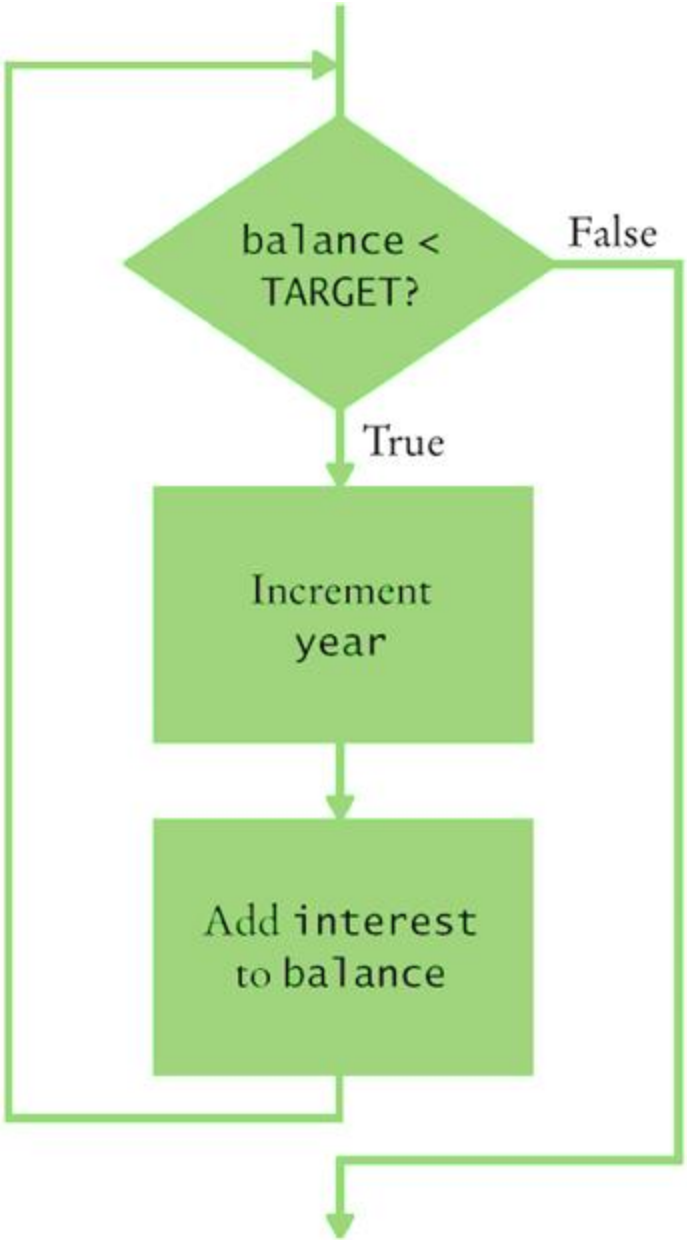
Execute the statement following the loop

balance = 20789.28

year = 15

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
cout << year << endl;
```

Flowchart of the Investment Calculation's `while` Loop



Example of a Problem – An Infinite Loop

`while` loop

```
i = 5;
while (i > 0)
{
    cout << i << " ";
    i++;
}
```

The output never ends

```
5 6 7 8 9 10 11...
```

Another Normal Execution – No Errors

```
i = 5;
while (i > 5)
{
   cout << i << " ";
   i--;
}
```

What is the output?

Another Normal Execution – No Errors

`while` loop

```
i = 5;
while (i > 5)
{
   cout << i << " ";
   i--;
}
```

There is (correctly) no output

The expression `i > 5` is false initially, so the statements are never executed.

A Very Difficult Error to Find (especially after looking for it for hours and hours!)

```
i = 5;

while (i > 0);
{
   cout << i << " ";
   i--;
}
```

What is the output?

A Very Difficult Error to Find (especially after looking for it for hours and hours!)

Another infinite loop – caused by a single character:

That semicolon causes the `while` loop to have an "empty body" which is executed forever.

The `i` in `(i > 0)` is never changed.

`while` loop

```
i = 5;

while (i > 0);
{
    cout << i << " ";
    i--;
}
```

There is no output!

The `for` Loop

C++ has a statement custom made *for* this sort of processing:

the `for` loop.

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

1 Initialize counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

2 Check counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

3 Execute loop body

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

4 Update counter

counter = 2

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

5 Check counter again

counter = 2

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

The `for` Statement

SYNTAX 4.2 for Statement

These three expressions should be related.

This *initialization* happens once before the loop starts.

The loop is executed while this *condition* is true.

This *update* is executed after each iteration.

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

The variable `i` is defined only in this `for` loop.

This loop executes 6 times.

Example of Normal Execution

`for` loop to hand-trace

```
for (int i = 0;
     i <= 5;
     i++)
   cout << i << " ";
```

What is the output?

Example of Normal Execution

`for` loop

```
for (int i = 0;
     i <= 5;
     i++)
   cout << i << " ";
```

The output

```
0 1 2 3 4 5
```

Note that the output statement is executed six times, not five

Example of Normal Execution – Taking Bigger Steps

`for` loop to hand-trace

```
for (int i = 0;
     i < 9;
     i += 2)
   cout << i << " ";
```

What is the output?

```
0 2 4 6 8
```

What is the output?

Infinite Loops Can Occur in `for` Statements

The danger of using == and/or !=

`for` loop to hand-trace

```
for (int i = 0;
     i != 9;
     i += 2)
   cout << i << " ";
```

What is the output?

Infinite Loops Can Occur in `for` Statements

`==` and `!=` are best avoided in the check of a `for` statement

`for` loop

```
for (int i = 0;
     i != 9;
     i += 2)
    cout << i << " ";
```

The output never ends

```
0 2 4 6 8 10 12…
```

Example of Normal Execution – Taking Even Bigger Steps

`for` loop to hand-trace

```
for (int i = 1;
     i <= 20;
     i *= 2)
   cout << i << " ";
```

What is the output?

The update can be any expression

Example of Normal Execution – Taking Even Bigger Steps

`for` loop

```
for (int i = 1;
     i <= 20;
     i *= 2)
   cout << i << " ";
```

The output

```
1 2 4 8 16
```

The "step" can be multiplicative or any valid expression

4.4 The do Loop

The `while` loop's condition test is the first thing that occurs in its execution.

The `do` loop (or `do-while` loop) has its condition tested only after at least one execution of the statements.

```
do
{
    statements
}
while (condition);
```

The do Loop

Here is the code:

```
int value;
do
{
   cout << "Enter a value < 100: ";
   cin >> value;
}
while (value >= 100);
```

In this form, the user sees the same prompt each time until the enter valid input.

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int n = 0;
7       do
8       {
9           n++;
10          cout<<n<<",";
11      }while(n<=10);
12      cout<<"end of the loop"<<endl;
13  }
```

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int n;
7       do
8       {
9           cout<<"Enter a number : ";
10          cin>>n;
11          cout<<"The number you entered : "<<n<<en
12      }while(n!=0);
13  }
```

4.7 Common Loop Algorithms

1: Sum and Average Value
2: Counting Matches
3: Finding the First Match
4: Maximum and Minimum
5: Comparing Adjacent Values

More Nested Loop Examples

The loop variables can have a value relationship. In this example the inner loop depends on the value of the outer loop.

```
for (i = 1; i <= 4; i++)
   {
for (j = 1; j <= i; j++)
      cout << "*";
cout << endl;
   }
```

The output will be:

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

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

```
int i = 0;
while (i < 10) {
  cout << i << "\n";
  i++;
   if (i == 4) {
    break;
  }
}
```

```
for (int i = 1; i <= weeks; ++i) {
        cout << "Week: " << i << endl;

        for (int j = 1; j <= days_in_week; ++j) {
            cout << "    Day:" << j << endl;
        }
    }
```

Output

```
Week: 1
    Day:1
    Day:2
    ... .. ...

Week: 2
Week: 3
    Day:1
    Day:2
    ... .. ...
```

Output

Write program for the following output

```
Week: 1
    Day:1
    Day:2
    Day:3
    ... .. ...



Week: 2
    Day:1
    Day:2
    Day:3
    ... ... ..
```

```
int weeks = 3, days_in_week = 7;

    for (int i = 1; i <= weeks; ++i) {
        cout << "Week: " << i << endl;

        for (int j = 1; j <= days_in_week; ++j) {
            // break during the 2nd week
            if (i == 2) {
                break;
            }
            cout << "    Day:" << j << endl;
        }
    }
```

Week: 1

Day:2

Day:4

Day:6

Week: 2

Day:2

Day:4

Day:6

Week: 3

Day:2

Day:4

```
int weeks = 3, days_in_week = 7;

    for (int i = 1; i <= weeks; ++i) {
        cout << "Week: " << i << endl;

        for (int j = 1; j <= days_in_week; ++j) {
            // continue if the day is an odd number
            if (j % 2 != 0) {
                continue;
            }
            cout << "    Day:" << j << endl;
        }
    }
```