C Looping Statements

The while statement

```
while(exp) statement
```

Operation:

The expression is evaluated. If it is "TRUE," the statement is executed, then the expresssion is evaluated again. If the expression is "FALSE" the statement is NOT executed, and control is passed to the statement following the while

while Examples

```
while(ch != ' ') cin >> ch;


k = 0;
while(k < 10) k = k + 1;


k = 0;
while(k < 10)
  {
   cin >> ch;
   k++;
   cout << ch;
  }  //  END while
cout << "You input " << k << "characters\n";
```

More while Examples

Input 10 values, compute and output the sum and average of the values

```
i = 0;
sum = 0.0;
while(i < 10)
  {
   cin >> val;
   sum += val;  // sum = sum + val;
   i++;
  }  // END while
avg = sum / float(i);
cout << "The sum of the values is " << sum << endl
     << "The average is " << avg << endl;
```


More while Examples

Input an unknown number of values, stopping when a 0 value is input, then compute and output the sum and average. ("Zero value termination")

```
i = 0;
sum = 0.0;
cin >> val;
while(val != 0.0)
  {
   sum += val;
   i++;
   cin >> val;
  }  // END while
avg = sum / float(i);
cout << "The sum and average of the inputs are "
     << sum << " and " << avg << endl;
```

C Looping Structures

The do-while statement

```
do statement while (exp);
```

Operation:

The statement is executed. Then the expression is evaluated. If the expression is "TRUE," the statement is executed again. If the statement is "FALSE," The statement following the do-while is executed, thereby terminating the loop.

This is a "post test" loop - the statement will always be executed at least once.

do-while Example

Zero value termination with a post-test loop.

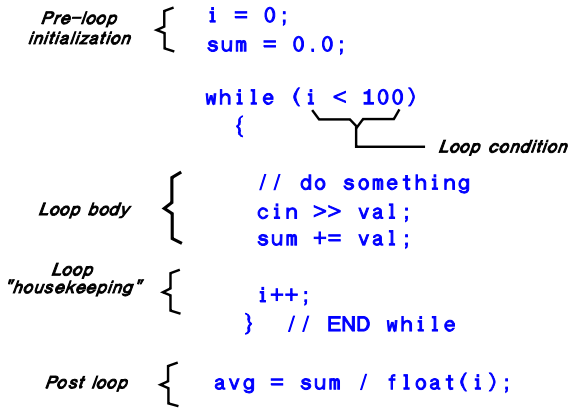
```
i = 0;
sum = 0.0;
do
  {
   cin >> val;
   sum += val;
   i++;
  }while(val != 0.0);
avg = sum / float(i-1); // Note the count adjustment
```

Another do-while Example

Input pairs of numbers and computer their sum. Continue as long as user continues to answer 'y'

```
do
  {
   cin >> a >> b;
   c = a + b;
   cout << c << endl;
   cout << "Continue? ";
   cin >> ans;
  } while(ans == 'y');
```


Anatomy of a Typical Loop



The C for Loop

"The swiss army knife of looping statements"

```
for( expr1; expr2; expr3 ) statement
```

Initialization — *Condition* — *End-of-loop Housekeeping*

Operation:

First, expr1 is performed (it is the pre-loop initialization). Then expr2 is evaluated; if TRUE, then statement is performed, then expr3 is executed and expr2 is evaluated again. If expr2 is FALSE, control goes to the statement following the for statement.

Comparing while and for

```
for( expr1; expr2; expr3 ) statement
```

Is exactly identical to the following while statement:

```
expr1;
while (expr2)
  {
   statement
   expr3;
  }  //END while
```

Loop Anatomy with for

```
for( expr1;   expr2;   expr3 ) statement
```

Pre-loop initialization — *Loop Condition* — *Loop "housekeeping"* — *Loop body*

University of Idaho

Some for Examples

```
sum = 0.0;
for( i = 0; i < 10; i++)
  {
   cin >> val;
   sum += val;  // sum = sum + val;
  }  // END for
avg = sum / float(i);
```

```
sum = 0.0;
cin >> val;
for(i=0; val != 0.0; i++)
  {
   sum += val;
   cin >> val;
  }  // END for
avg = sum / float(i);
```


University of Idaho

More for Examples – Good Practice?

```
cin >> val;
for(i=1, sum=val; val != 0.0; i++, sum+=val)
  {
   cin >> val;
  }  // END for
avg = sum / float(i-1);
```

```
cin >> val;
i=1; sum=val;
for( ; val != 0.0; )
  {
   cin >> val;
   i++; sum +=val;
  }  // END for
avg = sum / float(i-1
```

```
for(i=0,sum=0.0,cin>>val; val != 0.0; i++,sum+=val,cin>>val);
avg = sum / float(i);
```


Example – A Temperature Conversion Table

This code generates a centigrade to fahrenheit table, given the starting and ending points input by the use. It increments the temperatur by 10.

```
float fahr, cent, highC, lowC;

cout << "Enter starting and ending values: ";
cin >> lowC >> highC;

cout << "Centigrade to Fahrenheit Table\n";
cout << "Centigrade   Fahrenheit\n";

for( cent = lowC; cent <= highC; cent += 10)
  {
   fahr = cent * 9./5. + 32.;
   cout << "    " << cent << "          "
                  << fahr << endl;
  } // END for
```

Nested Loops

```
for(i = 0; i < 3; i++)
  {
   for(j = 0; j < 4; j++)
     {
      cout << i << j << endl;
     } // END for j
  } // End for i
```


Multiple Nesting

```
for(i = 0; i < 3; i++)
  {
   for(j = 0; j < 4; j++)
     {
       if (j < 2)
         {
          k = i + j;
          cout << k;
         } // END if
     } // END for j
  }// END for i
```

What is the output of this code?

Auxiliary Control Statements

```
break;
```

Causes an exit from the current program block. Execution continues at the statement following the block.

```
continue;
```

Causes the rest of the program block to be ignored, but does not exit the block. In loops, this means that the current iteration is terminated and the next one starts.

```
goto label;
```

Causes control to transfer to the statement with the specified label.

EX.

```
if (error) goto errlabel:
           .
           .
           .
errlabel: cout << "An error occured!\n";
```