Count Controlled Input

In some cases, the number of input records to be read will be specified in the input file itself. This is sometimes called the data header approach.

```
7
79
65
78
62
73
81
89
```

Here, we have a file of temperature data, with the first line specifying how many temperature values are given.

We might process this input file with the following loop:

```
. . .
   inData >> tempsPromised;   // get # temps expected
   inData >> Temperature;     // read first one

   while ( inData  && tempsRead < tempsPromised ) {

      tempsRead++;                  // count temps read
      // processing code goes here
      . . .

      inData >> Temperature;        // read next one
   }
. . .
```

Note that we do NOT place absolute trust in the data header value.

 ©1995-2009 Barnette ND, McQuain WD & Kennan MA

Complete Example

Let's extend the last code fragment to a complete program. Suppose that we are required to determine the highest temperature and the overall average temperature.

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

int main() {

   ifstream inData("Temperature.data");

   int    tempsPromised;         // # temp values expected
   int    tempsRead   = 0;       // # temps read so far
   int    hiTemp      = INT_MIN; // max temp so far
   double sumOfTemps = 0.0;      // sum for averaging
   int    Temperature;           // temperature just read

   inData >> tempsPromised;  // get # temps expected
   inData >> Temperature;    // read first one
. . .
```

```
 7
70
65
73
62
73
81
89
```


©1995-2009 Barnette ND, McQuain WD & Kennan MA

Complete Example

```
   . . .
   while ( inData  && tempsRead < tempsPromised ) {

      tempsRead++;                    // count temps read
                                      //   and keep sum
      sumOfTemps = sumOfTemps + Temperature;

      if ( Temperature > hiTemp )   // check for new max
         hiTemp = Temperature;      //   update if so

      inData >> Temperature;        // read next one
   }

   // Report results:
   if ( tempsRead > 0 ) {
      cout << "Highest temperature was:  "
           << hiTemp << '.' << endl;

      double avgTemp = sumOfTemps / tempsRead;
      cout << fixed << showpoint;
      cout << "Average temperature was:  "
           << setprecision(1) << avgTemp << '.' << endl;
   }

   return 0;
}
```

7
70
65
73
62
73
81
89

©1995-2009 Barnette ND, McQuain WD & Kennan MA

End-of-line Controlled Loop


```
. . .
   In.get(Next);     // Read 1st char

   while ( In ) {
      lineLength = 0;            // Restart line length count.
      while (Next != '\n') {     // Process next line:
         lineLength++;           //     count char
         cout << Next;           //     echo char
         In.get(Next);           // Read next char.
      }
      numChars = numChars + lineLength;
      numLines++;
      cout << setw(50 - lineLength) << lineLength << endl;
      In.get(Next);
   }

   cout << "Number of characters: " << numChars << endl;
   cout << "Number of lines:      " << numLines << endl;
. . .
```

```
Things should be made¶
as simple as possible,¶
but no simpler.¶§
```

¶	represents the newline char
§	represents the end of file char

©1995-2009 Barnette ND, McQuain WD & Kennan MA

Loop Design Considerations

Questions that one should consider carefully when coding a loop:

- What is the condition that terminates the loop?
- How should the condition be initialized?
- How should the condition be updated?
- What guarantees this condition will eventually occur?
- What is the process to be repeated?
- How should the process be initialized?
- How should the process be updated?
- What is the state of the program on exiting the loop?
- Are "boundary conditions" handled correctly?

©1995-2009 Barnette ND, McQuain WD & Kennan MA

`for` Loop

For loops are used whenever the number of times a loops needs to execute is known or can be calculated beforehand.

```
for (initial expression; test expression; update expression)
    <statement>
```

The initial expression is performed just once, prior to loop execution. The initial expression may declare variables as well as specify initializations for them.

The test expression is evaluated and if false then the next statement after the for loop is executed.

If the test expression evaluates to true, then the <statement> of the for loop is executed, the update expression is performed, and test expression is evaluated again.

For loops are generally count-controlled, but hybrid control is also fairly common, especially when input failure control is involved.

©1995-2009 Barnette ND, McQuain WD & Kennan MA

`for` Loop Example

Example:

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

int main() {

   const char ASTERISK = '*';                                          //  1
   int numAst;                                                         //  2
   int numPrinted;                                                     //  3

   cout << "How many asterisks do you want?  ";                        //  4
   cin  >> numAst;                                                     //  5

   for (numPrinted =0; numPrinted < numAst; numPrinted++) {            //  6

      cout << ASTERISK;                                                //  7
   }
   cout << endl;                                                       //  8

   return 0;                                                           //  9
}
```

Compare this with the implementation on slide 3 using a `while` loop.

The `for` loop is simply an alternative control structure (to the `while`); any `for` loop can be expressed as a `while` loop, and vice versa.

©1995-2009 Barnette ND, McQuain WD & Kennan MA

More `for` Loop Examples


```
const char Separator  = '-';
const int Length = 80;
int toWrite;

for (toWrite = Length; toWrite > 0; toWrite--) { // count down
   cout << Separator;
}
```

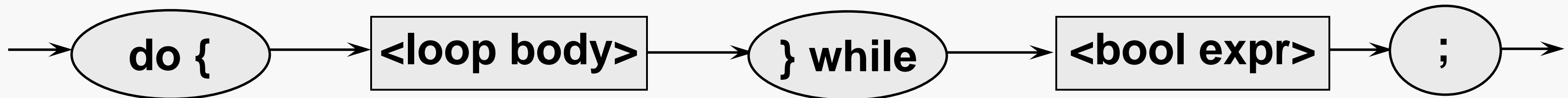
```
const int LIMIT = 100;
int sumOfSquares,
    Curr;

for (Curr = 1, sumOfSquares = 0; Curr <= LIMIT; Curr++)
    sumOfSquares = sumOfSquares  +  Curr * Curr;

cout << "Sum of squares = " << sumOfSquares;
```

```
. . .
for (Curr = 1, sumOfSquares = 0;
     Curr <= LIMIT;
     sumOfSquares = sumOfSquares  +  Curr * Curr, Curr++)
   ;  // empty body, all the "action" is in the update section
. . .
```


©1995-2009 Barnette ND, McQuain WD & Kennan MA

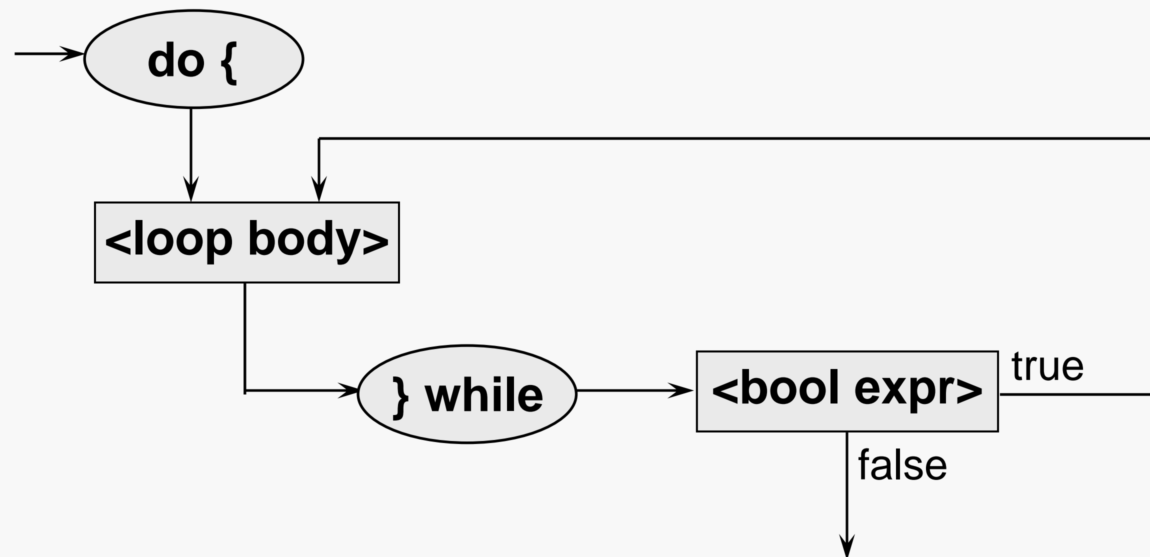
do-while Loop

The third loop construct in C++ is the do-while statement. Syntactically:



The Boolean expression must be enclosed in parentheses, and **<loop body>** can be either a single statement or a list of statements. A peculiarity is that the loop body must be enclosed in braces, even if it is a single statement.

Semantically:



©1995-2009 Barnette ND, McQuain WD & Kennan MA

do-while Example

Example: find first occurrence, if any, of a specific character in an input stream:

```
const char TOFIND = 'X';
char Next;
int  Skipped = -1;

do {
   In.get(Next);
   Skipped++;
} while (In && Next != TOFIND);

if ( In ) {
   cout << TOFIND " found after " << Skipped
        << " characters were skipped." << endl;
}
else {
   cout << TOFIND " was not found." << endl;
}
```


©1995-2009 Barnette ND, McQuain WD & Kennan MA