Chapter 6

Looping

By Dale, Weems, & Headington
With Modifications by M. L. Malone
March 08

Outcomes. At the end of this chapter students will know how to…

- Design and implement while loops
  - count-controlled
  - event-controlled loops
    - Using end of file condition
    - Using sentinels
- Design and implement nested loops
- Test and debug loops

What is a loop?

- A repetition control structure.
  a.k.a. an "iterative" structure
- Causes a single statement or block to be executed repeatedly

Two Types of Loops

count controlled loops

Repeat a specified number of times
Controlled by a counter

event-controlled loops

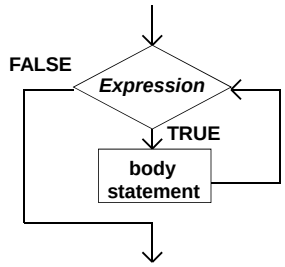
Some condition within the loop body changes and this causes the repetition to stop

While Loop

SYNTAX

```
while ( Expression )
 {      .
        .          // loop body
        .
 }
```


WHILE LOOP



Count-controlled Loop

- Loop control variable must be initialized
- Must have expression to test for continuing the loop
- Loop control variable updated with each iteration of the loop body

7

Count-controlled Loop

```
int   count ;

count  =  4;                        // initialize loop variable

while (count > 0)                   // test expression
{
        cout  << count  << endl ;   // repeated action
        count - - ;                 // update loop variable
}
cout  << "Done" << endl ;
```

8

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count

OUTPUT

9

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 4

OUTPUT

10

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)    TRUE
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 4

OUTPUT

11

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 4

OUTPUT: 4

12

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 3

OUTPUT:
4

13

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)    TRUE
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 3

OUTPUT:
4

14

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 3

OUTPUT:
4
3

15

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 2

OUTPUT:
4
3

16

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)    TRUE
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 2

OUTPUT:
4
3

17

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 2

OUTPUT:
4
3
2

18

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 1

OUTPUT:
4
3
2

19

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)    TRUE
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 1

OUTPUT:
4
3
2

20

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 1

OUTPUT:
4
3
2
1

21

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 0

OUTPUT:
4
3
2
1

22

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)    FALSE
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 0

OUTPUT:
4
3
2
1

23

Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
        cout  << count  << endl ;

        count - - ;
}
cout  << "Done" << endl ;
```

count: 0

OUTPUT:
4
3
2
1
Done

24

Modify it so that it loops 4 times

```
int   count ;

count  =  4;

while (count   ___??____)
{
        cout  << count  << endl ;
        count ++ ;
}
cout  << "Done" << endl ;
```

25

Count-Controlled Loop Example

myInfile contains 100 blood pressures

Use a while loop to read the 100 blood pressures and find their total.

Do this now!

(see template on next slide)

26

```
ifstream    myInfile ;
int             thisBP ;
int             total ;
int             count ;

count =  ___;                                   // initialize?

// Loop 100 times exactly!
while   ( ____)                         // test expression?
{
            myInfile   >>  thisBP ;
            total = total + thisBP ;
            ____                                // update count?

}
cout  <<  "The total = "  <<  total  <<  endl ;
```

One possible answer…

```
ifstream    myInfile ;
int             thisBP ;
int             total ;
int             count ;

count = 0 ;                                     // initialize

while   ( count < 100 )                 // test expression
{
            myInfile   >>  thisBP ;
            total = total + thisBP ;
            count++ ;                           // update
}
cout  <<  "The total = "  <<  total  <<  endl ;
```

Another possible answer…

```
ifstream    myInfile ;
int             thisBP ;
int             total ;
int             count ;

count = 1 ;                                     // initialize

while   ( count <= 100 )                // test expression
{
            myInfile   >>  thisBP ;
            total = total + thisBP ;
            count++ ;                           // update
}
cout  <<  "The total = "  <<  total  <<  endl ;
```

Design an algorithm using a count-controlled loop to…

- Average n integers
- User enters n at keyboard
- User then enters the integers at the keyboard
- n is a positive integer

Pseudo-code or C++ OK!

Do this now!

30

One possible answer…

prompt for n
read in n
sum ← 0
count← 0;
while (count < n)
    prompt/read for next integer (num)
    sum ← sum + num
    count ← count + 1
write out sum/n

31

Event-Controlled Loop

- **An event takes place that causes the loop to terminate**
- **Several types**
  - ***End-of-file controlled***
  - **Sentinel controlled**
  - **Flag controlled**

32

```
// End-of-file controlled loop
total = 0;
myInfile  >>  thisBP;   // priming the read !!!
while (myInfile)          // while last read successful
{
        total = total + thisBP;
        myInfile  >>  thisBP;    // read another
}
cout  <<  total;
```

33

```
// VERSION II: End-of-file controlled loop
total = 0;
myInfile  >>  thisBP;   // priming the read !!!
while (!myInfile.eof()) // while last read successful
{
        total = total + thisBP;
        myInfile  >>  thisBP;    // read another
}
cout  <<  total;
```

34

See "FileProcessing2" C++ program from "Handouts" Web page

- It shows…
  - Processing while not end of file
  - Checking for a missing file
  - Checking for an empty file

35

```
//End-of-file at keyboard
total = 0;
cout  << "Enter blood pressure (Ctrl-Z to stop)";
cin  >> thisBP;       // priming read
while (cin)           // while last read successful
{
        total = total + thisBP;
        cout  << "Enter blood pressure";
        cin >> thisBP;     // read another
}
cout << total;
```

36

Event-controlled Loops

- **End-of-file controlled** ✔
  keep processing data as long as there is more data in the file
- **Flag controlled**
  keep processing data until the value of a flag changes in the loop body
- **Sentinel controlled**
  keep processing data until a special value which is not a possible data value is entered to indicate that processing should stop

Using Different Types of Loops for the Blood Pressure Problem

Count controlled loop	✔ **Read exactly 100 blood pressures from a file.**
End-of-file controlled loop	✔ **Read all the blood pressures from a file no matter how many are there.**

Using Different Types of Loops for the Blood Pressure Problem continued…

Sentinel controlled loop	**Read blood pressures until a special value (like -1) selected by you is read.**
Flag controlled loop	**Read blood pressures until a dangerously high BP (200 or more) is read.**

A Sentinel-controlled Loop

- What is a "sentinel"?
- Requires "priming the read" (as was needed for reading "while not end of file"

40

```
// Sentinel controlled loop
total = 0;                          Why prime the read?
cout  <<  "Enter a blood pressure (-1 to stop ) ";
cin  >> thisBP;

while (thisBP != -1)  // while value read not sentinel
{
        total = total + thisBP;
        cout <<  "Enter a blood pressure (-1 to stop ) ";
        cin >> thisBP;
}
cout << total;
```

Flag-controlled Loops

- Uses a Boolean flag
- Must initialize to true or false
- Needs meaningful name
- Condition in the loop body changes the flag's value
- Must test for the flag in the loop test expression

42

Algorithm to count only the good blood pressure Readings & terminate if bad reading encountered

```
countGoodReadings = 0;
isSafe = true;     // initialize Boolean flag

while (isSafe)     // same as "while (isSafe==true)
{
        cin >> thisBP;
        if ( thisBP >= 200 )
                isSafe = false;      // change flag value
        else
                countGoodReadings++;
}
cout << countGoodReadings << endl;
```

43

Another example of a flag-controlled loop

```
// Play a game as long as user wants to play

bool keepPlaying = true;
char response;
while (keepPlaying)
   {
        // code here to play a game
        …

        cout << "Play again? y/n : ";
        cin >> response;
        response = upcase(response);
        if (response=='N')
                keepPlaying = false
   }
cout << "Thanks for playing!";
```

Advantage of using the Boolean keepPlaying?

44

Other Uses of Loops

- Count all data values
- Count only special data values
- Sum data values
- Keep track of previous and current values

45

Keeping Track of Previous and Current Values

- Write a program that counts the number of "!=" operators in a program's source code
- Read one character in the file at a time
- Must keep track of current & previous characters

46

Keeping Track of Values

```
(x != 3)
{
    cout << endl;
}
```

FILE CONTENTS

previous	current	count
(	x	0
x	' '	0
' '	!	0
!	=	1
=	' '	1
' '	3	1
3	)	1

```
int     count;
char  previous;
char  current;

count = 0 ;
inFile.get (previous);
inFile.get(current);          // prime the read

while (!inFile.eof())
{     if ( (previous ==  '!') && (current  ==  '=') )
            count++;
      previous = current;  // update
      inFile.get(current);   // read next character
}
```

48

Nested Loop

- **Loop within a loop**

49

Pattern of a Nested Loop

```
initialize outer loop
while  ( outer loop condition )
{      . . .
       initialize inner loop
       while ( inner loop condition )
       {
              inner loop processing and update
       }
        . . .
}
```

– ***How can you hide this detail (of loops inside of loops) in your program design?***

51

To design a nested loop

- **Begin with outer loop**
- **When you get to where the inner loop appears, *make it a separate module and come back to its design later***

Example, next slide→

52

Use a Nested Loop to Display Rows, Columns of $s

```
read in numRows, numCols
rowCtr ← 0
colCtr← 0
while (rowCtr<= numRows)
    while (colCtr <= numCols)
            write out "$"
            colCtr ← colCtr + 1
    rowCtr ← rowCtr + 1
    write newline character
```

Output for numRows = 4, numCols = 3

```
$$$
$$$
$$$
$$$
```

53

Display rows & columns of "$"s

```
read in numRows, numCols
rowCtr ← 0
colCtr← 0
while (rowCtr<= numRows)
    Display one row of "$"s
    rowCtr ← rowCtr + 1
    write newline character
```

Output for numRows = 4 numCols = 3

```
$$$
$$$
$$$
$$$
```

54

Best! Using a Subroutine

```
read in numRows, numCols
rowCtr ← 0
while (rowCtr<= numRows)
    DisplayRow(numCols)
    rowCtr ← rowCtr + 1
    write newline character
```

Output for numRows = 4 numCols = 3

```
$$$
$$$
$$$
$$$
```

55

Design for the Subroutine DisplayRow

Subroutine: DisplayRow
Task: Displays one row of dollar signs
PreCondition: numCols has value <=80
PostCondition: numCols $s have been displayed in one row

Parameters:	name	type	param type	description
	numCols	integer	value(in)	number of Columns
Local variables:				
	colCtr	integer		column counter for loop

Algorithm:

```
colCtr=1
while (colCtr <= numCols)
    write out "$"
    colCtr ← colCtr + 1
```

56

Using DisplayRow in C++

```
// prototype
void DisplayRow (int);
…
int main()
{
        read in numRows, numCols
        rowCtr = 0
        while (rowCtr<= numRows)
        {
                DisplayRow (numCols);
                rowCtr = rowCtr + 1;
                cout << endl;
        }
        return 0;
}
```

57

The Procedure Itself (follows the "main" module in C++ program)

```
// DisplayRow prints out one row of dollar signs
// PreCondition:        numCols has value <=80
// PostCondition:       numCols $s have been displayed in one row
//-----------------------------------------------------------------------------
void DisplayRow (int numCols)
{
        int colCtr=1;     // column counter for loop

        while (colCtr <= numCols)
        {
                cout << '$';
                colCtr = colCtr + 1;
        }
}
```

58

Another Example: Patient Data

A file (input) contains blood pressure data for different people. Each line has a patient ID, the number of readings for that patient, followed by the actual readings.

ID	howMany	Readings				
4567	5	180	140	150	170	120
2318	2	170	210			
5232	3	150	151	151		

59

Patient Date continued…

Read the data and display the average blood pressures for each patient

Patient ID	BP Average
4567	152
2318	190
5232	151
.	.
.	.
.	.

There were 432 patients in file.

60

Pseudocode Algorithm

initialize patientCount to 0
read first ID and howMany from file
while not end-of-file
    increment patientCount
    display ID
    loop to sum up this patients BPs
    calculate and display average for patient
    read next ID and howMany from file
display patientCount

61

```
#include <iostream>
#include <fstream>

using namespace std;

int main ( )
{
    int         patientCount;
    int         thisID;
    int         howMany;
    int         thisBP;
    int         totalForPatient;
    int         count;

    float       average;

    ifstream  myInfile;
```

```
myInfile.open("BP.dat");

if (!myInfile )   // opening failed
      cout << "File opening error. Program terminated.";
else                // file opened OK
{
    cout << "ID Number      Average BP" << endl;
    patientCount = 0;
    myInfile >> thisID >> howMany;        // priming read

    // loop to process patients BPs
              //see next slide for this loop
```

```
    while ( myInfile )    // while last read successful
    {
        patientCount++;
        cout << thisID;
        totalForPatient = 0;  // initialize inner loop counter
        count = 0;
        while ( count < howMany) // while still more to read
        {
                myInfile >> thisBP;
                count ++;
                totalForPatient = totalForPatient  + thisBP;
        }   // end while count<howMany

        average = totalForPatient / float(howMany);
        cout  <<  int (average + .5) << endl;  // round
        myInfile  >> thisID  >>  howMany;     // another read
    }   // end while myInfile
```

```
        cout   <<   "There were " <<   patientCount
               <<   "patients on file." << endl;

        cout  <<   "Program terminated.\n";
    }   // end else file opened OK

    return 0;
}
```

65

Notes re previous example…

- For loop would have been better for reading in blood pressures
  - Why?
- Comments at end of if-else clauses, loops, etc are optional but help reader tell what the closing brace "}" goes with

66

Tips on Loop Testing &Debugging

- Verify using an algorithm walk-through
- Trace execution of loop by hand with code walk-through
- Use a debugger to run program in “slow motion”
- Use test data to test all sections of program
- Beware infinite loops
- Check initial value of loop counter
- Check loop termination condition
  - Watch for “off-by-1” problem
- Check how loop counter modified in loop
- Use get function for loops controlled by detection of ‘\n’ character

67

Other loops?

- Read over Chapter 9
  - The do-while loop
  - The for-loop

68

What’s next?

- Read/study Chapters 7 & 8 Value-returning functions
  - Procedures (aka void functions in C++)
  - Scope of identifiers

69