ITERATIVE STATEMENTS

OVERVIEW

OVERVIEW

- **We often need to do repetitive calculations in order to solve specific problems**
  - Example: calculate the average GPA of all students at UofA
  - Humans are typically very slow and inaccurate doing this
  - Fortunately computers can do this quickly and correctly

- **To perform repetitive calculations in a program we need <u>iterative statements</u> that let us execute the same block of code multiple times**

OVERVIEW

- **C++ has three kinds of iterative statements**
  - The while loop
  - The for loop
  - The do-while loop

- **Lesson objectives:**
  - Learn the syntax and semantics of these iterative statements
  - Study example programs showing their use
  - Complete online labs on iterative statements
  - Complete programming project using iterative statements

ITERATIVE STATEMENTS

PART 1

WHILE LOOPS

WHILE LOOPS

- **A while loop iteratively executes a block of code**
- **We need to specify the following:**
  - The initialization code to execute before the loop
  - The logical expression for continuing iteration
  - The block of code to be repeated inside the loop
- **The program will execute block of code repeatedly as long as the while condition remains <u>true</u>**
- **The code directly after the loop is executed when the while condition becomes <u>false</u>**

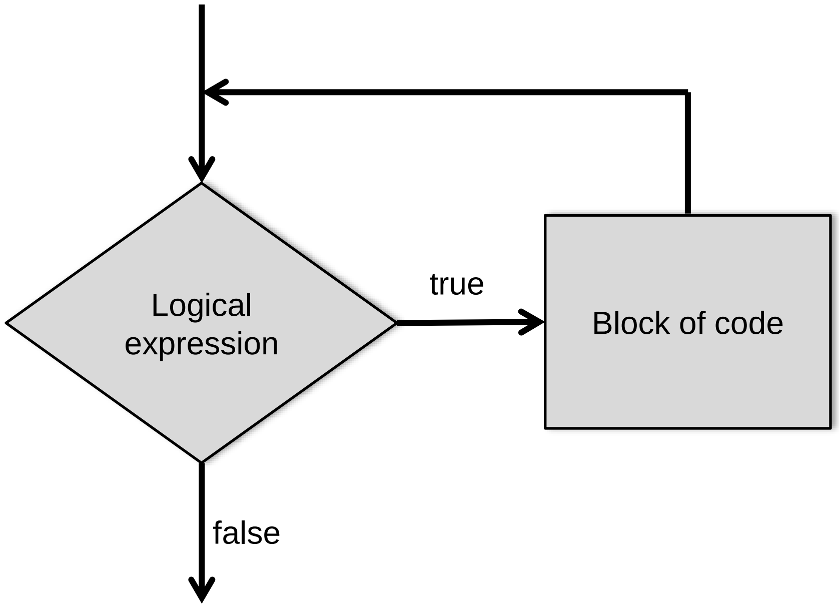
WHILE LOOPS

- **The C++ syntax of the while loop is:**

```
// initialization statement
while ( logical expression  )
{
   // block of statements to be repeated
   // update variables in logical expression
}
```

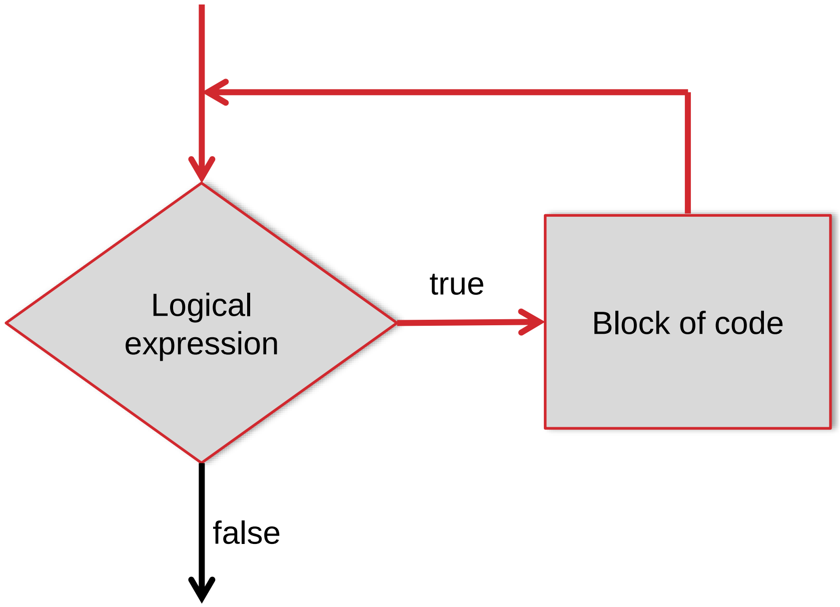
WHILE LOOPS

- **We can visualize the program's while loop decision process using a "flow chart" diagram**



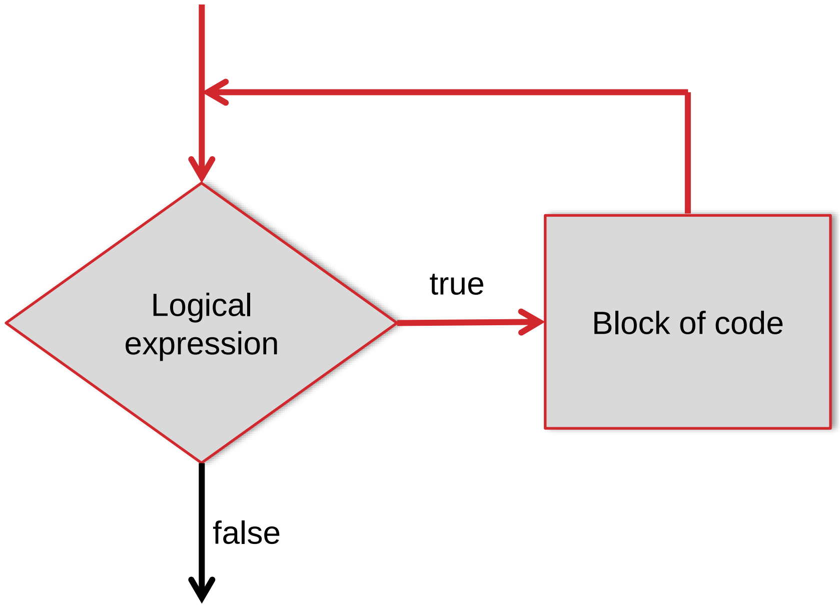
WHILE LOOPS

- If the logical expression is true, we take one path through the diagram (executing the block of code one time)



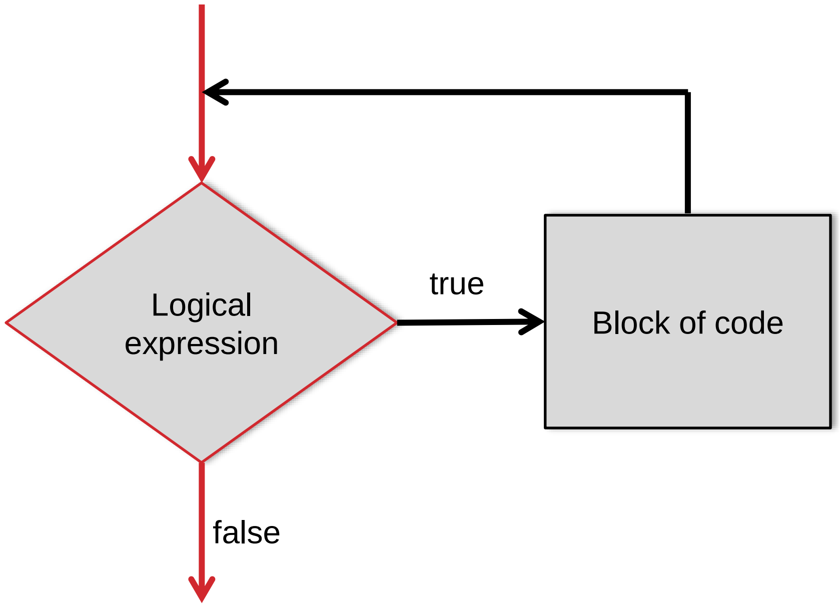
WHILE LOOPS

- **If the logical expression is still true, we follow the same path (executing the block of code again)**



WHILE LOOPS

- **When the logical expression is false, we take a different path through the diagram (skipping the block of code)**



COUNTING LOOPS

- **We can use a counting loop to perform some calculations a fixed number of times**

- **To do this we need to do the following:**
  - Initialize the loop counter
  - While counter has NOT reached desired value
    - Perform some calculations
    - Increment the loop counter
    - Check the loop counter again

COUNTING LOOPS

Counting loop example:

```
// Initialize counter
int Count = 0;
// Loop checking counter
while (Count < 10)
{
   // Perform some calculations
   cout << Count << " squared = " << Count*Count << endl;
   // Increment counter
   Count = Count + 1;
}
```

COUNTING LOOPS

Counting loop output:

```
0 squared = 0
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
5 squared = 25
6 squared = 36
7 squared = 49
8 squared = 64
9 squared = 81
```

COUNTING LOOPS

Some observations about this counting loop example:

- **The while loop will execute the cout statement 10 times and print Count values from 0,1,2,3,4,5,6,7,8,9**
- **At the bottom of the 10th iteration, we increment Count from 9 to 10, and the while condition becomes false, so the loop will stop executing**
- **After the while loop, the Count variable is equal to 10**

COUNTING LOOPS

Another counting loop example:

```
// Initialize counter
int Number = 1;
// Loop checking counter
while (Number <= 10)
{
   // Perform some calculations
   cout << Number << " halved = " << Number / 2 << endl;
   // Increment counter
   Number = Number + 1;
}
```

COUNTING LOOPS

Counting loop output:

```
1 halved = 0
2 halved = 1
3 halved = 1
4 halved = 2
5 halved = 2
6 halved = 3
7 halved = 3
8 halved = 4
9 halved = 4
10 halved = 5
```

Notice that we are doing integer division, so the fractional part is discarded 5 / 2 = 2 instead of 2.5

COUNTING LOOPS

Some observations about this counting loop example:

- **The while loop will execute the cout statement 10 times and print Number values from 1,2,3,4,5,6,7,8,9,10**
- **At the bottom of the 10th iteration, we increment Number from 10 to 11, and the while condition becomes false, so the loop will stop executing**
- **After the while loop, the Number variable is equal to 11**

COUNTING LOOPS

```
// Zero iterations loop
int Value = 11;
while (Value <= 7)
{
   cout << Value << " doubled = " << Value * 2 << endl;
   Value = Value + 1;
}
```

- **This while loop will execute the block of code zero times because the logical expression is false before loop starts**

COUNTING LOOPS

```
// User controlled counting loop
int Value = 0;
int StopValue = 0;
cin >> StopValue;
while (Value <= StopValue)
{
   cout << Value << " doubled = " << Value * 2 << endl;
   Value = Value + 1;
}
```

- **This program will read the value of StopValue from the user and will execute the while loop StopValue+1 times**

COUNTING LOOPS

Counting loop output:

```
5
0 doubled = 0
1 doubled = 2
2 doubled = 4
3 doubled = 6
4 doubled = 8
5 doubled = 10
```

This is the user input for StopValue

The cout statement in the while loop is executed StopValue+1 times

CONDITIONAL LOOPS

- **We often need to vary the number of loop iterations based on the values of one or more variables**
- **Conditional loops allow us to process data until given situation arises**

- **We need to do the following:**
  - Initialize condition variables
  - While the condition remains TRUE
    - Perform desired operations
    - Update condition variables

CONDITIONAL LOOPS

```
// Conditional loop example
int Amt = 42;
while (Amt > 0)
{
   cout << Amt << " halved = " << Amt / 2 << endl;
   Amt = Amt / 2;
}
```

- **This conditional loop will divide the Amt variable by 2 over and over again until Amt becomes equal to zero**

CONDITIONAL LOOPS

Conditional loop output:

```
42 halved = 21
21 halved = 10
10 halved = 5
5 halved = 2
2 halved = 1
1 halved = 0
```

Amt = 0 after this is printed, so the conditional loop will stop

CONDITIONAL LOOPS

```
// Another conditional loop example
int Val = 54;
int Cnt = 0;
while ((Val % 3) == 0)
{
   cout << "Val: " << Val << " Cnt: " << Cnt << endl;
   Val = Val / 3;
   Cnt = Cnt + 1;
}
 cout << "Val: " << Val << " Cnt: " << Cnt << endl;
```

This statement is true if Val is evenly divided by 3

- **This conditional loop will calculate the number of times that the number 3 is a factor of Val**

CONDITIONAL LOOPS

Conditional loop output:

```
Val: 54 Cnt: 0
Val: 18 Cnt: 1
Val: 6 Cnt: 2
Val: 2 Cnt: 3
```

After the conditional loop finishes, Cnt=3 tells us the number of times that 3 is a factor of 54

CONDITIONAL LOOPS

- **One special case for conditional loops is to read and process data from the user until they enter a sentinel value to signal the end of the input**

- **The basic approach is to:**
  - Prompt user for desired input
  - Read first input value from user
  - While input value is not the sentinel value
    - Process the input value
    - Read next input value from user

CONDITIONAL LOOPS

```
// Input varying loop
int Num = 0;
cin >> Num;
while (Num >= 0)
{
   float Val = sqrt(Num);
   cout << Num << " square root = " << Val << endl;
   cin >> Num;
}
```

We read input just before while loop and also in last line of loop

- **This loop will process a sequence of input values until the user enters the sentinel value (-1) to stop the loop**

CONDITIONAL LOOPS

Conditional loop input and output:

```
16 ←————————————————————— The user input
16 square root = 4 ←————— The program output
9
9 square root = 3
42
42 square root = 6.48074
-1 ←————————————————————— The conditional loop stops when
                          the user enters the sentinel value
```

ERROR CHECKING LOOPS

- **In many programs we ask the user to enter a data value within some specified range**
- **We can use while loops to perform error checking on the user input, and keep looping until correct data is entered**

- **The basic approach is to:**
  - Prompt user for desired input
  - Read input value from user
  - While value is NOT correct
    - Print error message and ask for input again
    - Read input value from user

ERROR CHECKING LOOPS

```
// Error checking loop example
cout << “Enter value between 17 and 42\n”;
cin >> Value;
while ((Value < 17) || (Value > 42))
{
   // Print error message and read another input
   cout << “Error:  Please enter value between 17 and 42\n”;
   cin >> Value;
}
cout << “Value = “ << Value << endl;
```

- **This error checking loop will execute zero or more times and only stop when the user input is valid**

ERROR CHECKING LOOPS

Error checking loop output:

```
Enter value between 17 and 42
11
Error:  Please enter value between 17 and 42
-22
Error:  Please enter value between 17 and 42
21
Value = 21
```

In this case, the user entered several invalid values before entering a valid value to stop loop

INFINITE LOOPS

- **It is possible to create while loops which execute forever**
  - These infinite loops are often unplanned and unwanted
- **To get out of infinite loop you need to kill your program**
  - Hit control-C on linux system
- **Occasionally infinite loops are used on purpose**
  - This is not recommended, but you may see it in other programmer's code

INFINITE LOOPS

```
// Infinite loop example
while (true)
   cout << "Hello Mom\n";
```

- **This while loop will print "Hello Mom" on the screen in an infinite loop until you kill the program by hitting control-C**

INFINITE LOOPS

```
// Accidental infinite loop
int Total = 0;
int Count = 0;
while (Count < 10)
{
   Total = Total + Count;
   cout << "total=" << Total << endl;
}
```

- **We forgot to increment the variable Count inside the loop so it will always be equal to 0, giving us an infinite loop**

INFINITE LOOPS

```
// Potential infinite loop
int Height = 0;
while (Height < 42)
{
   cout << "Enter height: ";
   cin >> Height;
}
```

- **This loop will execute over and over until the user enters a Height value >= 42**
- **This code could go in an infinite loop if the user types a string like “height” instead an integer value**

SUMMARY

- In this section we have introduced the syntax and use of the C++ while loop
- We also demonstrated several counting while loops and conditional while loops
- Finally, we discussed the problem of infinite loops, how to detect them, and how to avoid them

ITERATIVE STATEMENTS

PART 2

FOR LOOPS

FOR LOOPS

- **The C++ for loop provides a compact syntax for iteration**
- **For loops are typically used for counting loops, but they can be used for any looping task**

- **Allows you to specify the following all on one line**
  - Initialization statement
  - Logical expression for continuing loop
  - Statement to be executed after loop

FOR LOOPS

- **The C++ syntax of the for loop is:**

```
for ( initialization; logical expression; increment)
{
   // block of statements to be repeated
}
```

FOR LOOPS

```
// For loop example
for ( int Num = 0; Num < 10; Num = Num+1 )
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
}
```

- **The initialization statement is executed first**

FOR LOOPS

```
// For loop example
for ( int Num = 0; Num < 10; Num = Num+1 )
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
}
```

- **Then the logical expression is evaluated**

FOR LOOPS

```
// For loop example
for ( int Num = 0; Num < 10; Num = Num+1 )
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
}
```

- **If logical expression is <u>true</u> the block of code is executed**
- **If it is <u>false</u>, the program skips over the block of code**

FOR LOOPS

```
// For loop example
for ( int Num = 0; Num < 10; Num = Num+1 )
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
}
```

- **Next the increment statement is executed**

FOR LOOPS

```
// For loop example
for ( int Num = 0; Num < 10; Num = Num+1 )
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
}
```

- **Then the logical expression is evaluated again**
- **If it is <u>true</u>, we execute the block of code, the increment statement and the logical expression again**
- **If it is <u>false</u>, the for loop ends**

FOR LOOPS

```
// While loop example
int Num = 0;
while (Num < 10)
{
   cout << Num << "cubed=" << Num*Num*Num << endl;
   Num = Num+1;
}
```

- **This while loop does the same thing as the for loop above**

FOR LOOPS

```
// Another for loop example
for (int Amt = 42; Amt > 0; Amt = Amt/2)
{
   cout << Amt << " halved=" << Amt/2 << endl;
}
```

- **This for loop does the same work as one of our while loop examples from the previous lesson**
  - This code is several lines shorter but it is also harder to understand than a while loop implementation

FOR LOOPS

```
// Input varying loop
int Num = 0;
for (cin >> Num;  Num >= 0; cin >> Num)
    cout << Num << " square root=" << sqrt(Num) << endl;
```

- **This for loop does the same work as one of our while loop examples from the previous lesson**
  - This code is several lines shorter because the initialization, logical expression, and increment are all on one line
  - Since the for loop body is only one line long, we omitted the curly brackets { } to save two more lines of code

FOR LOOPS

```
// Infinite loop example
for (int Val = 1; Val > 0; Val = Val+1)
{
   cout << Val << " doubled=" << Val*2 << endl;
}
```

- **This is an example of an accidental infinite loop**
  - The value of the Val variable will always be greater than zero so the logical expression will always be true
  - The program will print values to the screen until the user kills the program by typing control-C

CONVERTING LOOPS

- **Computationally, for loops and while loops are identical**
  - The only real difference is in the C++ syntax
  - Some programmers prefer compact syntax of for loops
  - Other programmers prefer the simplicity of while loops

- **It is easy to convert a while loop into a for loop**
  - Change “while” to “for”
  - Move the initialization statement to start of for loop line
  - Keep logical expression the same
  - Move block of statements into for loop body
  - Move increment statement to end of for loop line

CONVERTING LOOPS

```
// initialization
while ( logical expression  )
{
   // block of statements to be repeated
   // increment
}

for ( initialization; logical expression; increment)
{
   // block of statements to be repeated
}
```

Copy each part of the <u>while</u> loop into the corresponding part of the <u>for</u> loop

CONVERTING LOOPS

```
// initialization
while ( logical expression )
{
  // block of statements to be repeated
  // increment
}

for ( initialization; logical expression; increment)
{
  // block of statements to be repeated
}
```

Copy each part of the while loop into the corresponding part of the for loop

CONVERTING LOOPS

```
// initialization
while ( logical expression  )
{
  // block of statements to be repeated
  // increment
}

for ( initialization; logical expression; increment)
{
  // block of statements to be repeated
}
```

Copy each part of the while loop into the corresponding part of the for loop

CONVERTING LOOPS

```
// initialization
while ( logical expression  )
{
  // block of statements to be repeated
  // increment
}

for ( initialization; logical expression; increment)
{
  // block of statements to be repeated
}
```

Copy each part of the while loop into the corresponding part of the for loop

CONVERTING LOOPS

- **It is also easy to convert a for loop into a while loop**
  - Change “for” to “while”
  - Move the initialization statement before while loop
  - Keep logical expression the same
  - Move block of statements into while loop body
  - Move increment statement to bottom of while loop body

CONVERTING LOOPS

```
for ( initialization;  logical expression;  increment)
{
   // block of statements to be repeated
}

// initialization
while ( logical expression  )
{
   // block of statements to be repeated
   // increment
}
```

Copy all four parts of the for loop into the four corresponding parts of the while loop

AUTO INCREMENT AND DECREMENT

- **The auto increment operator "++" can be used to quickly add one to an integer variable**
  - Instead of using i = i+1 we can use i++

- **The auto decrement operator "--" can be used to quickly subtract one from an integer variable**
  - Instead of using j = j-1 we can use j--

- **These operators are used quite frequently in for loops, especially counting loops to save typing**

AUTO INCREMENT AND DECREMENT

- **The auto increment and decrement operations can also be placed <u>before</u> the variable to add or subtract one**
  - There is a very subtle difference between ++i and i++
  - “cout << ++i” will add one to i, and then print i
  - “cout << i++” will print i, and then add one to i

AUTO INCREMENT AND DECREMENT

- **It is also possible to combine arithmetic operators with the assignment operator to save typing (and improve speed)**
  - We can replace a = a + b with a += b
  - Similarly c = c - d can be written as c -= d
  - The operators *=, /=, %= are also supported
  - This results in shorter and faster code
  - This syntax is also a little hard to read

AUTO INCREMENT AND DECREMENT

```
// Example using compact operators
int Sum = 0;
int Product = 1;
for (Count = 0;  Count < 13; Count++)
{
   Sum += Count;
   Product *= Count;
}
```

This is the same as
Count = Count + 1;

This is the same as
Sum = Sum + Count;
Product = Product * Count;

SUMMARY

- In this section we have introduced the syntax and use of the C++ for loop
- We also described how you can convert a for loop into a while loop and vice versa
- Finally, we described the C++ auto increment and auto decrement operators and related operations that combine an arithmetic operation with assignment

ITERATIVE STATEMENTS

PART 3

MORE LOOPS

DO WHILE LOOPS

- **In addition to while loops and for loops, C++ has another iterative statement called the do while loop**
  - Unlike other loops, the do while loop puts the logical expression <u>after</u> the body of loop
  - The body of loop will be always executed at least once
  - If logical expression is <u>true</u>, the loop will execute again
  - The do while loop ends when the expression is <u>false</u>
  - Do while loops are useful for limited number of applications

DO WHILE LOOPS

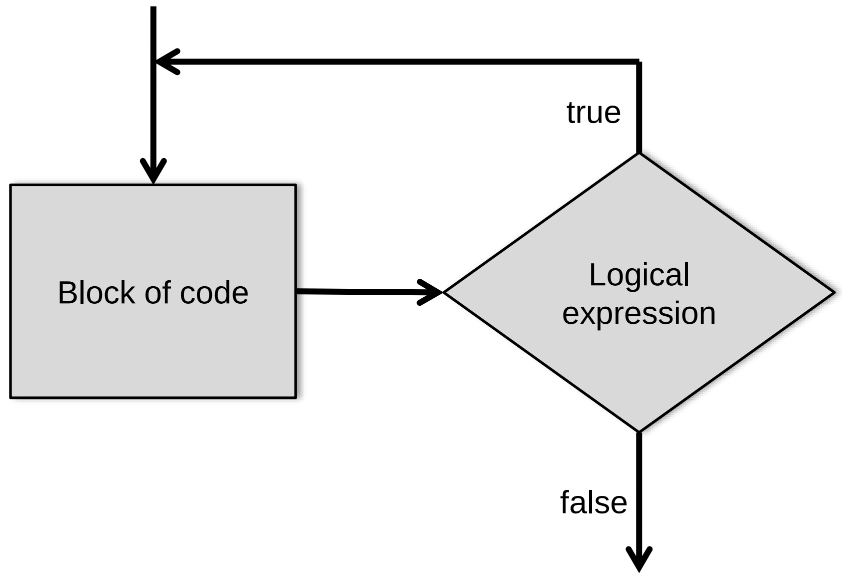
- **The C++ syntax of the do while loop is:**

```
do
{
   // block of statements to be repeated
}
while ( logical expression );
```

Notice that there IS a semicolon at the end of the while() line

DO WHILE LOOPS

- **We can visualize the program's do while loop decision process using a "flow chart" diagram (notice that the block of code is executed before the logical expression)**



DO WHILE LOOPS

```
// Do while example
int Value = 0;
do
{
   cout << "Enter number between [0..9] ";
   cin >> Value;
}
while ((Value < 0) || (Value > 9));
```

This do while loop will prompt the user for data one or more times until the correct value is entered

DO WHILE LOOPS

```
// Do while example
int Value = 0;
cout << "Enter number between [0..9] ";
cin >> Value;
while ((Value < 0) || (Value > 9))
{
   cout << "Enter number between [0..9] ";
   cin >> Value;
}
```

NESTED LOOPS

- **It is often necessary for one loop to include another loop to solve a problem**
  - This is called a nested loop
  - Both loops need separate initializations, logical expressions, and increments

- **How many times will nested loops execute?**
  - If the outer loop executes N times
  - And the inner loop executes M times each time it is reached
  - Then inner block of code will be executed N x M times
  - This analysis extends to three or more nested loops

NESTED LOOPS

```
 // Outer loop
for (int Height=0; Height < 14; Height++)
{
// Inner loop
for (int Width=0; Width < 17; Width++)
  {
    // Operation
    cout << "*";
  }
  cout << endl;
}
```

The outer loop executes 14 times

The inner loop executes 14*17 times

NESTED LOOPS

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

Sample output from the nested loop example above has 14 rows and 17 columns of *'s

NESTED LOOPS

```
 // Outer loop
for (int Number = 1; Number <= 15; Number++)
{
   // Inner loop
   int Factorial = 1;
   for (int Count = 1; Count <= Number; Count++)
   {
      // Operation
      Factorial = Factorial * Count;
   }
   cout << " Number = " << Number
         << " Factorial = " << Factorial << endl;
}
```

The outer loop executes 15 times

The inner loop executes Number times

NESTED LOOPS

```
Number = 1 Factorial = 1
Number = 2 Factorial = 2
Number = 3 Factorial = 6
Number = 4 Factorial = 24
Number = 5 Factorial = 120
Number = 6 Factorial = 720
Number = 7 Factorial = 5040
Number = 8 Factorial = 40320
Number = 9 Factorial = 362880
Number = 10 Factorial = 3628800
Number = 11 Factorial = 39916800
Number = 12 Factorial = 479001600
Number = 13 Factorial = 1932053504
Number = 14 Factorial = 1278945280
Number = 15 Factorial = 2004310016
```

These errors are caused by integer overflow

NESTED LOOPS

```
// Outer loop
for (int Number = 1; Number < 15; Number++)
{
  // Inner loop
  long Factorial = 1;
  for (int Count = 1; Count <= Number; Count++)
  {
    // Operation
    Factorial = Factorial * Count;
  }
  cout << " Number = " << Number
       << " Factorial = " << Factorial << endl;
}
```

We can fix the problem by using a long variable to store the Factorial value

NESTED LOOPS

```
Number = 1 Factorial = 1
Number = 2 Factorial = 2
Number = 3 Factorial = 6
Number = 4 Factorial = 24
Number = 5 Factorial = 120
Number = 6 Factorial = 720
Number = 7 Factorial = 5040
Number = 8 Factorial = 40320
Number = 9 Factorial = 362880
Number = 10 Factorial = 3628800
Number = 11 Factorial = 39916800
Number = 12 Factorial = 479001600
Number = 13 Factorial = 6227020800
Number = 14 Factorial = 87178291200
Number = 15 Factorial = 1307674368000
```

By using a long variable the values of 13! 14! and 15! are now correct

PRIME NUMBER EXAMPLE

- **Consider the problem of checking if a number is prime**
  - We need to see if it has any factors besides 1 and itself
  - Loop over all possible factors for number
  - The number is prime if no factor is found

- **How can we find <u>all</u> prime numbers less than 1000?**
  - Loop over all numbers from 1..1000
    - Loop over all possible factors for number
    - The number is prime if no factors is found

- **Nested loops will be needed to solve this problem**

PRIME NUMBER EXAMPLE

- **Top down approach**
  - Start by writing outer loop that goes from 1..1000
  - Debug program
  - Then fill in the inner loop to check for prime numbers
  - Debug program

- **Bottom up approach**
  - Start with inner loop to check for prime numbers
  - Debug program
  - Write the outer loop that goes from 1..1000
  - Debug program

PRIME NUMBER EXAMPLE

```
#include <cmath>
#include <iostream>
using namespace std;
int main()
{
   // Loop over range of values
   for (int Number = 2; Number < 1000; Number++)
   {
      bool Prime = true;
      // Add prime checking code here later
      if (Prime) cout << Number << " ";
   }
}
```

First, we write the outer loop that goes over a range of values we want to check for primes

PRIME NUMBER EXAMPLE

Initial program output:

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106
107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197 198 199 200 … 990 991 992 993 994 995 996 997 998 999
```

PRIME NUMBER EXAMPLE

```
#include <cmath>
#include <iostream>
using namespace std;
int main()
{
   // Loop over range of values
   for (int Number = 2; Number < 1000; Number++)
   {
      bool Prime = true;
      for (int Factor = 2; Factor <= sqrt(Number); Factor++)
         if ((Number > Factor) && (Number % Factor == 0))
            Prime = false;
      if (Prime) cout << Number << " ";
   }
}
```

Then we add the inner loop to check all possible factors up to square root of number and set Prime to false if a factor is found

PRIME NUMBER EXAMPLE

Final program output:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163
167 173 179 181 191 193 197 199 211 223 227 229 233 239 241
251 257 263 269 271 277 281 283 293 307 311 313 317 331 337
347 349 353 359 367 373 379 383 389 397 401 409 419 421 431
433 439 443 449 457 461 463 467 479 487 491 499 503 509 521
523 541 547 557 563 569 571 577 587 593 599 601 607 613 617
619 631 641 643 647 653 659 661 673 677 683 691 701 709 719
727 733 739 743 751 757 761 769 773 787 797 809 811 821 823
827 829 839 853 857 859 863 877 881 883 887 907 911 919 929
937 941 947 953 967 971 977 983 991 997
```

SOFTWARE ENGINEERING TIPS

- **Print debugging messages inside each loop**
  - So you know how many iterations have been executed
  - So you know what values your variables contain
- **Make sure the loop executes correct number of times**
  - Off by one errors are the most common
- **Anticipate loops that may execute zero times**
  - Make sure that subsequent code operates properly
- **Anticipate and avoid infinite loops**
  - Make sure you get closer to the terminating condition of the loop on each loop iteration

SOFTWARE ENGINEERING TIPS

- **Common programming mistakes**
  - Never update for loop counter variable inside for loop
  - Never use the same counter variable for nested loops
  - Missing or unmatched ( ) brackets in logical expressions
  - Missing or unmatched { } brackets in iterative statement
  - Never use & instead of && in logical expressions
  - Never use | instead of || in logical expressions
  - Never use = instead of == in logical expressions
  - Never use “;” directly after the for() or while() line

SUMMARY

- In this section we have studied the syntax and use of the C++ do while loop
- We also showed several example nested loops to create more complex iterative programs
- Finally, have discussed several software engineering tips for creating and debugging iterative programs