



# *Chapter 6: Repetition Statements*

Mr. Dave Clausen  
La Cañada High School



# *Repetition Statements*

- ◆ Our third control structure: iteration or repetition (completes our 3 control structures: sequence, selection, iteration)
- ◆ Two main categories of repetition:
  - ◆ definite loop
    - ◆ repeats a predetermined number of times
  - ◆ indefinite loop
    - ◆ repeats a number of times that has not been predetermined.



# *Repetition Forms*

- ◆ Three loop types:

- ◆ for<a definite number of times> <do action>
- ◆ while<condition is true> <do action>
- ◆ do<action> while <condition is true>

- ◆ Three basic constructs

- ◆ A variable is assigned some value.
- ◆ The value of the variable changes at some point in the loop.
- ◆ The loop repeats until the variable reaches a predetermined value, the program then executes the next statement after the loop.



# *Pretest Loops*

- ◆ Pretest Loop (Entrance Controlled Loops)
  - ◆ a loop where the control condition (Boolean expression) is tested BEFORE the loop.
  - ◆ If the condition is true, the loop is executed.
  - ◆ If the condition is false the loop is not executed
  - ◆ Therefore, it is possible that these loops may not be executed at all (when the condition is False)
  - ◆ There are two pretest loops
    - ◆ for loop
    - ◆ while loop



# *Post Test Loops*

- ◆ Post Test Loops (exit-controlled loop)
  - ◆ a loop where the control condition (Boolean expression) is tested **AFTER** the loop has been executed.
  - ◆ If the condition is true, the loop is executed again.
  - ◆ If the condition is false the loop is not executed again
  - ◆ Therefore, this type of loop will always be executed at least once.
  - ◆ There is one post test loop: `do...while`



# *Fixed repetition loops*

- ◆ Fixed repetition loop

- ◆ a loop used when you know in advance how many repetitions need to be executed.

- ◆ also known as a definite loop:

- ◆ ( you know the definite number of repetitions necessary to solve the problem)

- ◆ the “for” loop is:

- ◆ a fixed repetition loop

- ◆ and a pretest loop



# *Variable Condition Loops*

## ◆ Variable Condition Loops

- ◆ needed to solve problems where the conditions change within the body of the loop.
- ◆ Also called indefinite loops:
  - ◆ the loop repeats an indefinite number of iterations until some condition is met, or while some condition is met.
  - ◆ The loop terminates depending upon conditions involving sentinel values, Boolean flags, arithmetic expressions, end of line, or end of file markers.
  - ◆ While and do...while loops are variable condition loops.



# *The for Loop*

## ◆ General form:

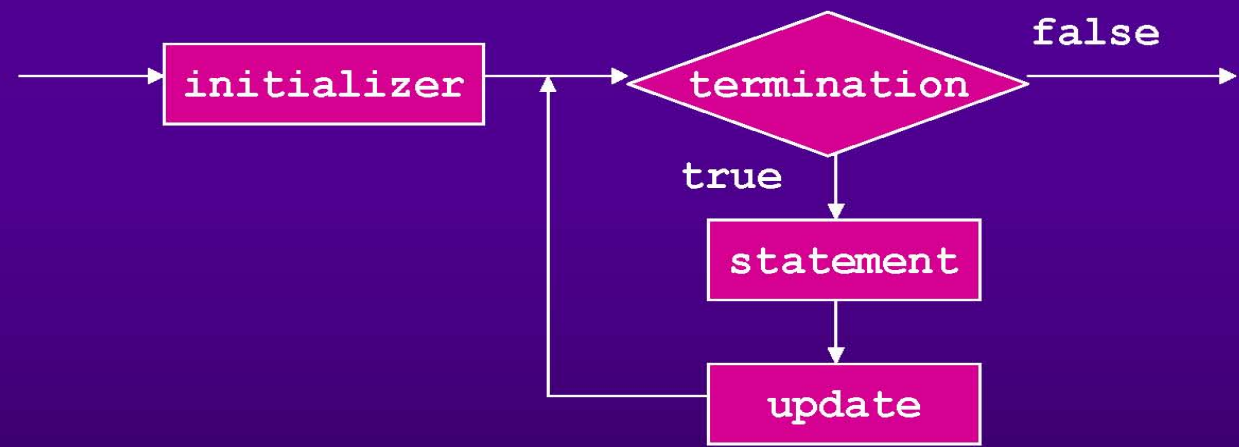
```
for(<initialization expression>; <termination conditon>; <update expression> )  
    <statement>
```

```
for (counter = 1; counter <= 10; counter = counter+1)    //Loop Heading  
    cout<< counter << endl;                            //Loop body
```

# *Syntax and Semantics of the **for** Loop*

`for (<initializer>; <termination>; <update>)`  
`<statement>`

Loop header                      Loop body





## *The for Loop Internal Logic*

- ◆ The control variable is assigned an initial value in the initialization expression
- ◆ The termination condition is evaluated
- ◆ If termination condition is true
  - ◆ the body of the loop is executed and the update expression is evaluated
- ◆ If the termination condition is false
  - ◆ program control is transferred to the first statement following the loop.



# *Increment Operator*

- ◆ The Increment operator adds 1 to the variable
- ◆ Instead of  $x = x + 1$  you can write as  $++x$ 
  - ◆ if the  $++$  occurs before the  $x$  ( $++x$ ) it is called a prefix operator
  - ◆ if the  $++$  occurs after the  $x$  ( $x++$ ) it is called a postfix operator
- ◆ Our text uses the prefix operator
  - ◆ But let's use the postfix operator,  $x++$



# *Decrement Operator*

- ◆ The Decrement operator subtracts 1 from the variable
- ◆ Instead of  $x = x - 1$  you can write as  $--x$ 
  - ◆ if the  $--$  occurs before the  $x$  ( $--x$ ) it is called a prefix operator
  - ◆ if the  $--$  occurs after the  $x$  ( $x--$ ) it is called a postfix operator
- ◆ Our text uses the prefix operator
  - ◆ But let's use the postfix operator,  $x--$



# Accumulator

- ◆ An accumulator is a variable used to keep a running total or sum of successive values of another variable
  - ◆ i.e.  $\text{sum} = \text{sum} + \text{grade}$ ;
  - ◆ you should initialize the value of the accumulator before the loop:  $\text{sum} = 0$ ;
  - ◆ the accumulator statement occurs in the body of the loop

*//counter --> means loop control variable*

```
sum=0;
```

```
for (counter = 1; counter <= 100; counter++)
```

```
    sum = sum + counter ;
```



# *Scope of Loop Control Variable*

- ◆ The loop control variable must be declared before it is used.
  - ◆ The rules for the scope of the variable apply here
- ◆ If the variable is only going to be used as a loop counter, and for nothing else...
  - ◆ You can limit it's scope by declaring it when it is initialized in the loop

```
for(int counter = 1; counter <=10; counter ++)  
    cout<< counter <<endl;  
// counter is only referenced in loop
```



# *For Loops*

- ◆ For loops can count down (decrement)  

```
for (int counter=20; counter>=15; counter--)  
    cout<< counter << endl;
```
- ◆ For loops can count by factors other than one  

```
for(int counter=2; counter<=10; counter=counter+2)  
    cout<< counter << endl;
```
- ◆ Style
  - ◆ Indent the body of the loop, use blank lines before and after, and use comments.



# *While Loops*

- ◆ General form:

```
while (<Boolean expression>
    <statement>
```

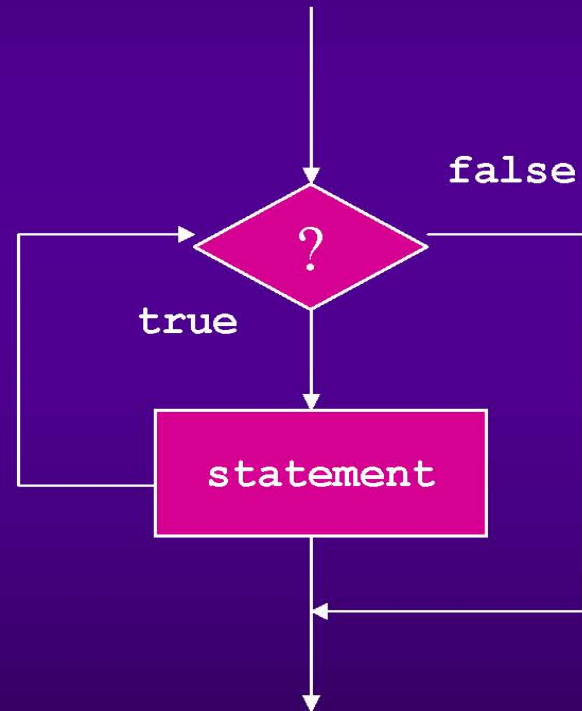
- ◆ The parentheses around the Boolean is required.
- ◆ If the condition is true the body of the loop is executed again.
- ◆ If the loop condition is false, the program continues with the first statement after the loop.
- ◆ A while loop may not be executed... why?



# Syntax and Semantics of *while* Statements

```
while (<Boolean expression>)  
  <statement>
```

```
while (<Boolean expression>)  
{  
  <statement 1>  
  .  
  <statement n>  
}
```





## *While Loops: Discussion*

- ◆ The condition can be any valid Boolean Expression
- ◆ The Boolean Expression must have a value **PRIOR** to entering the loop.
- ◆ The body of the loop can be a compound statement or a simple statement.
- ◆ The loop control condition needs to change in the loop body
  - ◆ If the condition is true and the condition is not changed or updated, an infinite loop could result.
  - ◆ If the condition is true and never becomes false, this results in an infinite loop also.



# *The while Loop Accumulator*

Write code that computes the sum of the numbers between 1 and 10.

```
int counter = 1;
int sum = 0;
while (counter <= 10)
{
    sum = sum + counter;
    counter = counter + 1;
}
```



# *Sentinel Values and Counters*

## ◆ Sentinel Value

- ◆ A value that determines the end of a set of data, or the end of a process in an indefinite loop.

[P309ex1.cpp](#)

[P309ex1.txt](#)

- ◆ While loops may be repeated an indefinite number of times.
  - ◆ It is common to count the number of times the loop repeats.
  - ◆ Initialize this “counter” before the loop
  - ◆ Increment the counter inside the loop

# *do...while loops*



◆ General form:

```
do
```

```
{
```

```
    <statement>
```

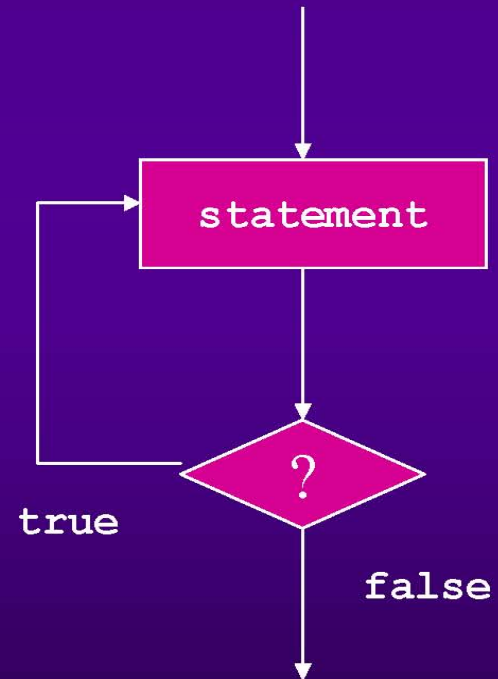
```
}while (<Boolean expression>)
```

- ◆ The Boolean expression must have a value before it is executed at the end of the loop.
- ◆ If the loop condition is true, control is transferred back to the top of the loop.
- ◆ If the loop condition is false, the program continues with the first statement after the loop.
- ◆ A do...while loop will always be executed at least once... why?

# Syntax and Semantics of *do...while* Statements

```
do
{
    <statement>
}while (<Boolean expression>);
```

```
do
{
    <statement 1>
    .
    <statement n>
} while (<Boolean expression>);
```





## *do...while Loops: Discussion*

- ◆ The condition can be any valid Boolean Expression
- ◆ The Boolean Expression must have a value PRIOR to exiting the loop.
- ◆ The body of the loop is treated as a compound statement even if it is a simple statement. { }
- ◆ The loop control condition needs to eventually change to FALSE in the loop body
  - ◆ If the condition never becomes false, this results in an infinite loop.



# *Choosing which loop to use.*

- ◆ for loop
  - ◆ when a loop is to be executed a predetermined number of times.
- ◆ while loop
  - ◆ a loop repeated an indefinite number of times
  - ◆ check the condition before the loop
  - ◆ a loop that might not be executed (reading data)
- ◆ do...while
  - ◆ a loop repeated an indefinite number of times
  - ◆ check the condition at the end of the loop



# *Designing Correct Loops*

- ◆ Initialize all variables properly
  - ◆ Plan how many iterations, then set the counter and the limit accordingly
- ◆ Check the logic of the termination condition
- ◆ Update the loop control variable properly



# *Off-by-One Error*

```
int counter = 1;
while (counter <= 10)
{
    // Executes 10 passes
    <do something>
    counter++;
}
```

```
int counter = 1;
while (counter < 10)
{
    // Executes 9 passes
    <do something>
    counter++;
}
```



# *Infinite Loop*

```
int counter = 1;
while (counter <= 10)
{
    // Executes 5 passes
    <do something>
    counter = counter + 2;
}
```

```
int counter = 1;
while (counter != 10)
{
    // Runs forever
    <do something>
    counter = counter + 2;
}
```

In general, avoid using `!=` in loop termination conditions.



## *Testing Loops*

- ◆ Can vary the limit or the control variable, or both
- ◆ Use a negative value, zero, and a positive value
- ◆ Display an output trace if things aren't working



# *Error Trapping*

```
//”primed” while loop
cout<<"Enter a score between " <<low_double<<" and " <<high_double;
cin>>score;
while((score < low_double) || (score > high_double))
{
    cout<<"Invalid score, try again.";

    //update the value to be tested in the Boolean Expression

    cout<<"Enter a score between " <<low_double<<" and
    "<<high_double;
    cin>>score;
}
```



# *Nested Loops*

- ◆ Nested loop

- ◆ when a loop is one of the statements within the body of another loop.

```
for (k=1; k<=5; ++k)
```

[Multab.cpp](#)

```
    for (j=1; j<=3; ++j)
```

[Multab.txt](#)

```
        cout<<(k+j)<<endl;
```

- ◆ Each loop needs to have its own level of indenting.
- ◆ Use comments to explain each loop
- ◆ Blank lines around each loop can make it easier to read



# *Repetition and Selection*

- ◆ The use of an if statement within a loop to look for a certain condition in each iteration of the loop.
  - ◆ Examples:
    - ◆ to generate a list of Pythagorean Triples
    - ◆ to perform a calculation for each employee
    - ◆ to find prime numbers
      - ◆ let's look at our Case Study program for Chapter 6

[primes.cpp](#)

[primes.txt](#)



# *Loop Verification*

- ◆ Loop verification
  - ◆ making sure that the loop performs its intended job.
- ◆ Input assertions
  - ◆ preconditions stating what is true before the loop is executed
- ◆ Output assertions
  - ◆ post conditions stating what is true after the loop is executed



# *Invariant and variant assertions*

## ◆ Loop Invariant

- ◆ states a relationship among variables that remains the same throughout all repetitions of the loop.
- ◆ A statement that is true both:
  - ◆ before the loop is entered, and
  - ◆ after each iteration of the loop

## ◆ Loop Variant

- ◆ an assertion that changes between the first and last iterations of the loop
- ◆ should be stated in a way that guarantees that the loop is exited.
- ◆ Should address the loop variable being incremented or decremented