

# CSC180: Lecture 7

Wael Aboulsaadat

wael@cs.toronto.edu

<http://portal.utoronto.ca/>

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Iteration

# Loops

- Can you
  - Write a function that prints the square of each of the first 100 numbers (1..99)
  - Write a function that computes the factorial of an integer

# do-while loop

- A variation of the while loop.
- A do-while loop is always executed at least once
  - The body of the loop is first executed
  - The Boolean expression is checked after the body has been executed

## Syntax of the *do-while* Statement

---

### A Loop Body with Several Statements:

```
do  
{  
    Statement_1  
    Statement_2  
    ...  
    Statement_Last  
} while (Boolean_Expression);
```

*body*

*Do not forget the  
final semicolon.*

### A Loop Body with a Single Statement:

```
do  
    Statement  
while (Boolean_Expression);
```

*body*

```
while (Boolean_Expression);
```

---

## Syntax of the *while* Statement and *do-while* Statement

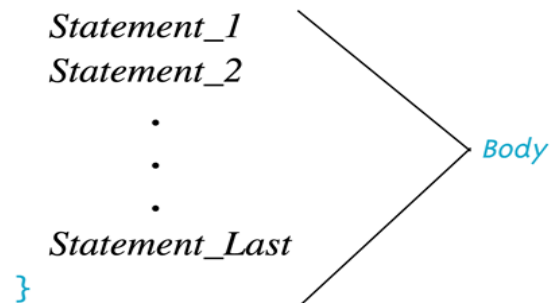
---

### A *while* Statement with a Single Statement Body

```
while (Boolean_Expression)  
    Statement ← Body
```

### A *while* Statement with a Multistatement Body

```
while (Boolean_Expression)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
}
```

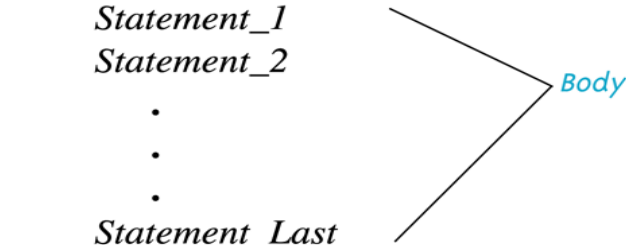


### A *do-while* Statement with a Single Statement Body

```
do  
    Statement ← Body  
while (Boolean_Expression);
```

### A *do-while* Statement with a Multistatement Body

```
do  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
} while (Boolean_Expression);
```



# The for-Statement

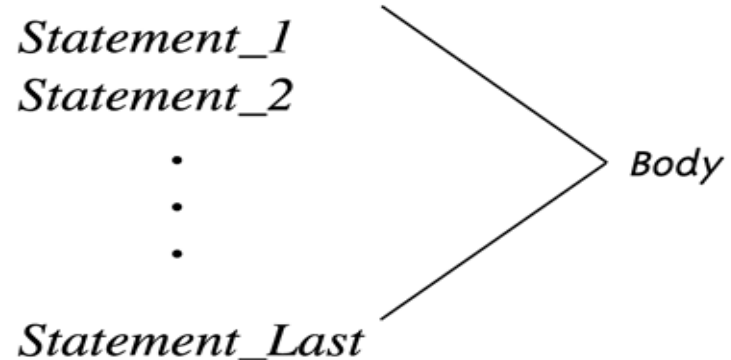
- A for-Statement (for-loop) is another loop mechanism in C
  - Designed for common tasks such as adding numbers in a given range
  - Is sometimes more convenient to use than a while loop
  - Does not do anything a while loop cannot do

## ***for* Loop with a Multistatement Body**

---

### **Syntax**

```
for (Initialization_Action; Boolean_Expression; Update_Action)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
}
```



A diagram illustrating the structure of the loop body. A large right-pointing chevron shape is formed by two lines. The top line starts at the right side of the opening curly brace and extends to the right, ending at the word "Body". The bottom line starts at the right side of the closing curly brace and extends to the right, also ending at the word "Body". The word "Body" is positioned to the right of the vertex of the chevron. Inside the chevron, the text "Statement\_1", "Statement\_2", three dots, and "Statement\_Last" are arranged vertically, representing the multistatement body of the loop.



# For Loop Dissection

- The for loop uses the same components as the while loop in a more compact form

- for (n = 1; n <= 10; n = n+1)



# for/while Loop Comparison

- `sum = 0;`  
`n = 1;`  
`while(n <= 10) // add the numbers 1 - 10`  
`{`  
`sum = sum + n;`  
`n = n + 1;`  
`}`
- `sum = 0;`  
`for (n = 1; n <= 10; n = n + 1) //add the numbers 1 - 10`  
`sum = sum + n;`

# for Loop Alternative

- A for loop can also include a variable declaration in the initialization action
  - `for (int n = 1, x = 10, y = 20; n <= 10; n = n + 1, x = x + 10)`

This line means

- Create a variable, `n`, of type `int` and initialize it with 1
- Continue to iterate the body as long as `n <= 10`
- Increment `n` by one after each iteration

# for-loop Details

- Initialization and update actions of for-loops often contain more complex expressions
  - Here are some samples

```
for (n = 1; n <= 10; n = n + 2)
```

```
for(n = 0 ; n > -100 ; n = n -7)
```

```
for(double x = pow(y,3.0); x > 2.0; x = sqrt(x) )
```

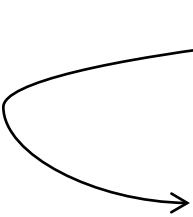
# The break-Statement

- There are times to exit a loop before it ends
  - If the loop checks for invalid input that would ruin a calculation, it is often best to end the loop
- The break-statement can be used to exit a loop before normal termination
  - Be careful with nested loops! Using break only exits the loop in which the break-statement occurs

# Example

```
int BreakTest( int breakvalue )
{
    int loopcounter = 0;

    while (loopcounter < 100)
    {
        if (loopcounter == breakvalue)
            break;
        loopcounter = loopcounter + 1;
    }
    return loopcounter;
}
```



# Designing Loops

# Designing Loops

- Designing a loop involves designing
  - The body of the loop
  - The initializing statements
  - The conditions for ending the loop



# Sums and Products

- A common task is reading a list of numbers and computing the sum
  - Pseudocode for this task might be:

```
sum = 0;  
repeat the following this_many times  
    getNextNumber( );  
    sum = sum + next;  
end of loop
```
  - This pseudo-code can be implemented with a for-loop as shown on the next slide

## for-loop for a Sum

- The pseudo-code from the previous slide is implemented as

```
int sum = 0;
for(int count=1; count <= this_many; count=count+1)
{
    next = getNextNumber( count );
    // getNextNumber maintains a list of numbers
    // and returns the number whose index is count
    sum = sum + next;
}
```

- sum must be initialized prior to the loop body!

# for-loop For a Product

- Forming a product is very similar to the sum example seen earlier

```
int product = 1;
for(int count=1; count <= this_many; count = count + 1)
{
    next    = getNextNumber( count );
    // getNextNumber maintains a list of numbers
    // and returns the number whose index is count

    product = product * next;
}
```

- product must be initialized prior to the loop body
- Notice that product is initialized to 1, not 0!

# Ending a Loop

- There are four common methods to terminate an input loop
  - List headed by size
    - When we can determine the size of the list beforehand
  - Ask before iterating
    - Ask if the user wants to continue before each iteration
  - List ended with a special value
    - Using a particular value to signal the end of the list

# List Headed By Size

- The for-loops we have seen provide a natural implementation of the list headed by size method of ending a loop

- Example:

```
int items, number;
items = getListSize( );
for(int count = 1; count <= items; count = count + 1)
{
    number = getNumberbyIndex( count );
    // getNumberbyIndex maintains a list of numbers
    // and returns the number whose index is count

    // statements to process the number
}
```

# Ask Before Iterating

- A while loop is used here to implement the ask before iterating method to end a loop

```
int sum = 0;
```

```
char ans = 'n';
```

```
ans = getUserAnswerToQuestion("Is there a new set of  
numbers?");
```

```
while (( ans == 'Y') || (ans == 'y'))
```

```
{
```

```
    //statements to read and process the numbers
```

```
    ans = getUserAnswerToQuestion("Is there a new set of  
numbers?");
```

```
}
```

# List Ended With a Special Value

- A while loop is typically used to end a loop using the list ended with a special value method

```
printf( "Enter a list of nonnegative integers.\n"  
        "Place a negative integer after the list \n");
```

```
int counter = 0;  
number = getNumberByIndex( 0 );  
while (number > 0)  
{  
    //statements to process the number  
  
    counter = counter + 1;  
    number = getNumberByIndex( counter );  
}
```

- Notice that the special value is read, but not processed