Do-While Loops; Loop Vocabulary

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Do-while loops
- Loop variables
  - Counter
  - Sentinel
  - Accumulator
- Loop types
  - Counter-controlled
  - Sentinel-controlled
  - User-controlled

While and If Statements

- Both skip code block entirely if condition is false
- `while` may execute multiple times if condition is true

```
int number = 2 + 5 / 3;
if(number < 3)
{
  Console.WriteLine("Hello!");
  Console.WriteLine(number);
  number++;
}
Console.WriteLine("Done");
```

```
int number = 2 + 5 / 3;
while(number < 3)
{
  Console.WriteLine("Hello!");
  Console.WriteLine(number);
  number++;
}
Console.WriteLine("Done");
```

Looping At Least Once

- What if you want to execute code at least once?
- `while` loop can require you to duplicate code

```
Console.WriteLine("Enter the item's price.");
decimal price = decimal.Parse(Console.ReadLine());
while(price < 0)
{
  Console.WriteLine("Invalid price. Please enter"
    + " a non-negative price.");
  price = decimal.Parse(Console.ReadLine());
}
Item myItem = new Item(desc, price);
```

Read and parse user input, store in `price`

Check value of `price`

Read and parse user input, store in `price`

The Do-While Loop

- Executes code block once, *then* evaluates condition
- If condition is true, repeat; if condition is false, proceed

```
decimal price;
do
{
  Console.WriteLine("Please enter a price "
    + "(must be non-negative).");
  price = decimal.Parse(Console.ReadLine());
} while(price < 0);
Item myItem = new Item(desc, price);
```

Declare `price` (scope is outside the loop)

Start loop block, but no condition yet

Read and parse user input, store in `price`

Check value of `price`

Just like `while` loop, `price < 0` must be false at this point

AUGUSTA UNIVERSITY

Do-While Syntax Details

- do keyword is required, but does nothing
- Semicolon required after `while` statement
- Condition is evaluated after executing loop block
- If true, go back to do, execute block again
- Curly braces can be omitted for single-statement loop blocks

```
do
{
   <statements>
} while(<condition>);
```

Same line as }

```
do
   <statement>
while(<condition>);
```

More Advanced Do-While Loop

- Ensure user input is a number, and a valid data value

```
decimal price;
string answer;
bool parseSuccess;
do
{
  Console.WriteLine("Please enter a price "
    + "(must be non-negative).");
  answer = Console.ReadLine();
  parseSuccess = decimal.TryParse(answer, out price);
} while(!parseSuccess || price < 0);
Item myItem = new Item(desc, price);
```

Both condition variables must be declared outside loop

Set to user's number, or 0

Loop again if TryParse failed

Loop again if price is negative

Outline

- Do-while loops
- **Loop variables**
  - Counter
  - Sentinel
  - Accumulator
- Loop types
  - Counter-controlled
  - Sentinel-controlled
  - User-controlled

Special Variable Vocabulary

- **Counter** variable: increases by 1 every time an event occurs
- In loops, usually counts the number of iterations

```
int counter = 0;
while(counter <= 3)
{
  Console.WriteLine("Hello again!");
  Console.WriteLine(counter);
  counter++;
}
Console.WriteLine("Done");
```

Named counter to be obvious

Also a counter variable

```
int i = 0;
while(i < 10)
{
  Console.WriteLine($"{i}");
  i++;
}
Console.WriteLine("Done");
```

Sentinel Values

- Sentinel value: A specific value (for a variable) that indicates execution should stop
- Causes the loop to end, rather than use/process the value

```
Console.WriteLine("Enter a string.");
string input = Console.ReadLine();
while(input != "quit")
{
  Console.WriteLine($"Your string was: {input}");
  Console.WriteLine("Enter another string, "
    + "or enter \"quit\" to quit.");
  input = Console.ReadLine();
}
```

Sentinel value is "quit"

Sentinel value will not be printed

AUGUSTA UNIVERSITY

Accumulators

- **Accumulator** variable: Maintains total of several values
- Increases by 1 or more each time a new value is added
- Usually not in loop condition, but modified by loop body

Note: i is still a counter

```
int i = 0, total = 0;   ← Accumulator is total
while(i < 10)
{
  total += i;   ← Add a new value to the accumulator
  i++;
}
Console.WriteLine($"The sum from 0 to {i} is {total}");
```

Combining Variable Types

- Before: Accumulator to compute total, counter to control loop
- Accumulator to compute total, sentinel to control loop:

```
int sum = 0;
Console.WriteLine("Enter a number to sum, or \"Done\" to stop");
string userInput = Console.ReadLine();
while(userInput != "Done")
{
   sum += int.Parse(userInput);
   Console.WriteLine("Enter a number to sum, or \"Done\" to stop");
   userInput = Console.ReadLine();
}
Console.WriteLine($"Your sum is: {sum}");
```

Accumulator is sum

Sentinel value is “Done”

AUGUSTA UNIVERSITY

Aside: Eliminating the Repetition

- Using a do-while loop and TryParse, no need for duplicate code

```
int sum = 0;
string userInput;
do
{
  Console.WriteLine("Enter a number to sum, or \"Done\" to stop");
  userInput = Console.ReadLine();
  int inputNum;
  int.TryParse(userInput, out inputNum);
  sum += inputNum;
} while(userInput != "Done");
Console.WriteLine($"Your sum is: {sum}");
```

Declare userInput outside the loop

If TryParse fails, this will be set to 0, which has no effect on sum

Combining All Three

- Sometimes you need counter, accumulator, and sentinel

```
int sum = 0, counter = 0; ← Accumulator is sum
Console.WriteLine("Enter a number to average, or \"D\" to stop");
string userInput = Console.ReadLine();
while(userInput != "D") ← Sentinel value is “D”
{
   sum += int.Parse(userInput); ← Add a new value to the accumulator
   counter++; ← Increment the counter
   Console.WriteLine("Enter a number to average, or \"D\" to stop");
   userInput = Console.ReadLine();
}
Console.WriteLine($"Your average is: {(double)sum / counter}");
```

Outline

- Do-while loops
- Loop variables
  - Counter
  - Sentinel
  - Accumulator
- **Loop types**
  - Counter-controlled
  - Sentinel-controlled
  - User-controlled

Counter-Controlled Loops

- Loop iteration controlled by counter variable
- Definite iteration: Number of iterations is known in advance
  - Even if it's not a constant

```
int i = 0;
while(i < 10)
{
   Console.WriteLine($"{i}");
   i++;
}
Console.WriteLine("Done");
```

Number of iterations = 10

```
int dollars = (int) myItem.GetPrice();
int i = 1;
while(i <= dollars)
{
   Console.WriteLine($"${i}");
   i++;
}
```

Number of iterations = whole dollars in price

AUGUSTA UNIVERSITY

Sentinel-Controlled Loops

- Loop iteration controlled by variable with sentinel value
- Indefinite iteration: Number of iterations not known

```
Console.WriteLine("Enter a positive number to continue, "
  + "or a negative number to stop.");
int input = int.Parse(Console.ReadLine());
while(input > 0)
{
  Console.WriteLine("Your number was: {input}");
  Console.WriteLine("Enter a positive number to continue, "
    + "or a negative number to stop.");
  input = int.Parse(Console.ReadLine());
}
```

Sentinel value: negative numbers

Sentinel-Controlled Loops

- Sentinel-controlled means sentinel is in loop condition, even if loop also has a counter or accumulator

```
int sum = 0;
Console.WriteLine("Enter a positive number to sum, "
  + "or a negative number when finished.");
int input = int.Parse(Console.ReadLine());
while(input > 0)
{
  sum += input;
  Console.WriteLine("Enter a positive number to sum, "
    + "or a negative number when finished.");
  input = int.Parse(Console.ReadLine());
}
 Console.WriteLine("Your total is: {sum}");
```

Sentinel

Accumulate

AUGUSTA UNIVERSITY

User-Controlled Loops

- Number of iterations depends on user input
- Can also be sentinel-controlled or counter-controlled

Loops as many times as necessary, until user's input is correct

```
int number;
do
{
  Console.WriteLine("Please enter a number "
    + "between 0 and 100.");
  number = int.Parse(Console.ReadLine());
} while(number < 0 || number > 100);
Console.WriteLine("Thank you! Your number "
  + $"is {number}");
```

User-Controlled Loops

- Number of iterations depends on user input
- Can also be sentinel-controlled or **counter-controlled**

```
Console.WriteLine("Enter a positive number.");
int numTimes = int.Parse(Console.ReadLine());
Console.WriteLine("Enter a string to print.");
string repeatString = Console.ReadLine();
int counter = 0;
while(counter < numTimes)
{
  Console.WriteLine(repeatString);
  counter++;
}
```

Counter-controlled: loop stops when counter reaches a known value

User-controlled: numTimes comes from user input

Summary

- Do-while loops
- Loop variables
  - Counter
  - Sentinel
  - Accumulator
- Loop types
  - Counter-controlled
  - Sentinel-controlled
  - User-controlled