Loops with Arrays and More Advanced Loops

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- For loops and arrays
  - Foreach loops
- Break and continue

Using Arrays

- Accessing array elements individually:

```
int[] homeworkGrades = {89, 72, 88, 80, 91};
double average = (homeworkGrades[0] + homeworkGrades[1] +
  homeworkGrades[2] + homeworkGrades[3] + homeworkGrades[4]) / 5.0;
```

- Can we write a for loop that does this with less repetition?

```
int sum = 0;
for(int i = 0; i < 5; i++)
{
  sum += homeworkGrades[i];
}
double average = sum / 5.0;
```

Size of array

Use <, stop when `i == 5`
(there is no `homeworkGrades[5]`)

Custom-Sized Arrays

- Array size can be any `int`, including a variable:

```
int numGrades = 10;
int[] homeworkGrades = new int[numGrades];
```

Creates an array of 10 `int`s

- Array size can be user-provided:

```
Console.WriteLine("How many grades are there?");
int numGrades = int.Parse(Console.ReadLine());
int[] homeworkGrades = new int[numGrades];
```

- How do we use this array when we don't know its size?

Custom-Sized Arrays

- for loops make it easy to process "the whole array"
  - End condition can be a variable: the size of the array

```
Console.WriteLine("How many grades are there?");
int numGrades = int.Parse(Console.ReadLine());
int[] homeworkGrades = new int[numGrades];
for(int i = 0; i < numGrades; i++)
{
  Console.WriteLine($"Enter grade for homework {i+1}");
  homeworkGrades[i] = int.Parse(Console.ReadLine());
}
```

Size of array

Homework 1 is in homeworkGrades[0]

Looping with the Length Property

- Arrays are objects with instance variables
- `int length` contains the length (size) of the array, can be accessed with property `Length`

```
class Array
{
  private int length;
  public int Length
  {
    get
    {
      return length;
    }
  }
}
```

Loops through all `int`s in `homeworkGrades`, however many

```
int sum = 0;
for(int i = 0; i < homeworkGrades.Length; i++)
{
  sum += homeworkGrades[i];
}
double average = (double) sum /
  homeworkGrades.Length;
```

No need for a counter

A Loop Shortcut

- `for` loops over arrays all look the same:

```
for(int i = 0; i < myArray.Length; i++)
{
  <do something with myArray[i]>;
}
```

- If you only need to **read** the array entries, a shorter form:

```
int sum = 0;
foreach(int grade in homeworkGrades)
{
  sum += grade;
}
```

Name of array

Variable to hold each value in array

Contains `homeworkGrades[0]`, then `homeworkGrades[1]`, etc.

foreach Rules

- Cannot be used to **change** values in array

```
foreach(int grade in homeworkGrades)
{
  grade = int.Parse(Console.ReadLine());
}
```

Error! Can't assign to grade

- Loop variable must match type of array:

```
string[] days = {"Mon", "Tue", "Wed", "Thu", "Fri"};
foreach(string day in days)
{
  Console.WriteLine(day);
}
```

Outline

- For loops and arrays
  - Foreach loops
- **Break and continue**

Conditional Loop Control

- What if you want to skip some iterations of the loop?
- Example: Only use even values from array, skip odd values

```
int sum = 0;
for(int i = 0; i < myArray.Length; i++)
{
  if(myArray[i] % 2 == 0)
  {
    Console.WriteLine(myArray[i]);
    sum += myArray[i];
  }
}
```

Check if current value is even

Entire loop body inside an `if` block

Skipping Iterations

- `continue` keyword = "skip this loop iteration"
- Return to loop beginning, increment counter, check condition

```
int sum = 0;
for(int i = 0; i < myArray.Length; i++)
{
  if(myArray[i] % 2 != 0)
    continue;
  Console.WriteLine(myArray[i]);
  sum += myArray[i];
}
```

Check if current value is **odd**

Immediately start next iteration

These are only executed if `myArray[i]` is even

Multiple End Conditions

- Scenario: Loop should end when a sentinel value is encountered, or when input is invalid

```
int sum = 0, userNum = 0;
bool success = true;
while(success && userNum >= 0)
{
  sum += userNum;
  Console.WriteLine("Enter a positive number to add it. "
    + "Enter anything else to stop.");
  success = int.TryParse(Console.ReadLine(), out userNum);
}
```

Extra variable to store parsing success

0 is a valid input, doesn't indicate failure

Another Way to End the Loop

- break keyword = "stop execution here" – ends the loop

```
int sum = 0, userNum = 0;
while(userNum >= 0)
{
  sum += userNum;
  Console.WriteLine("Enter a positive number to add it. "
    + "Enter anything else to stop.");
  if(!int.TryParse(Console.ReadLine(), out userNum)
    break;
}
```

Simpler condition, no variable needed

If TryParse failed, end the loop

Using break in a for Loop

- Scenario: Array is partially filled in with numbers, but at some (unknown) point, all the rest are zeroes


```
int product = 1;
for(int i = 0; i < myArray.Length; i++)
{
  if(myArray[i] == 0)
    break;
  product *= myArray[i];
}
```

Normal loop end: after last value in array

"Special" loop end: Encounter a zero

Summary

- For loops and arrays
  - Foreach loops
- Break and continue