Object Oriented Programming

Lecture 1 Introduction

Arsalan Rahman Mirza

Computer Science Department

Faculty of Science

2022-2023

Topics

- C# Variables and Types
- Conditions for C#, Java, and C++
- Boolean operators
- if – else statement
- Loops (while, for)
- Switch case
- Array
- Multidimensional Array
- List
- Dictionary

C# Variables and Types

- C# is a statically-typed language. Therefore, we must define the types of variables before using them.
- To define a variable in C#, we use the following syntax, which is similar to C / Java:

```
int myInt = 1;
float myFloat = 1f;
bool myBoolean = true;
string myName = "John";
char myChar = 'a';
double myDouble = 1.75;
```

Variables and Types

- C# supports type inference - which means that you don't always have to explicitly specify a type - you can let the compiler try and understand the type of variable automatically. However, once the type of variable has been determined, it cannot be assigned a different type.

```
var x = 1;
var y = 2;
var sum = x + y;
```

Conditions for C#, Java and C++

```
if ( condition )
{
   statement ;
}
```

Example

```
int a = 4;
if (a == 4)
{
   Console.WriteLine("Ohhh! So a is 4!");
}
```

Conditions

- C# uses Boolean variables to evaluate conditions. The Boolean values true and false are returned when an expression is compared or evaluated. For example:

```
int a = 4;

bool b = a == 4;

if (b)

{

    Console.WriteLine("It's true!");

}
```

Boolean operators

There aren't that many operators to use in conditional statements and most of them are pretty straightforward:

```
int a = 4;
int b = 5;
bool result;
result = a < b; // true
result = a > b; // false
result = a <= 4; // a smaller or equal to 4 - true
result = b >= 6; // b bigger or equal to 6 - false
result = a == b; // a equal to b - false
result = a != b; // a is not equal to b - true
result = a > b || a < b; // Logical or - true
result = 3 < a && a < 6; // Logical and - true
result = !result; // Logical not - false
```

if – else statement

- The if, else statement in C# is pretty simple.

```
if (a == b)
{
    // a and b are equal, let's do something cool
}
```

- And we can also add an else statement after an if, to do something if the condition is not true, The if, else statement in C# is pretty simple.

```
if (a == b)
{
    // We already know this part
}
else
{
    // a and b are not equal... :/
}
```

if – else statement

- The if - else statements doesn't have to be in several lines with {}, if can be used in one line, or without the {}, for a single line statement.

```
if (a == b)
    Console.WriteLine("Another line Wow!");
else
    Console.WriteLine("Double rainbow!");
```

- Although this method might be useful for making your code shorter by using fewer lines

if – else if statement

```
if (a == b)
{
      // We already know this part
}
else if
{
      // a and b are not equal... :/
}
.
.
.
.
else
{
      // something new
}
```

Loops (while)

- While loops are defined like this:

```
while ( condition )
{
    statement ;
}
```

- This allows you to continuously repeat a section of code while a condition is satisfied.

```
int n = 0;
while( n == 0)
{
    Console.WriteLine("N is 0");
    n++;
}
```

- This would execute once as N is changed from zero the first time it runs.

For loops

```
for ( initial;  condition ; action)
{
    statement ;
}
```

Example

```
int i;
for( i = 0; i < 10; i++)
{
    //code to execute
}
```

This would loop until i is no longer less than ten, increasing i by one each time. C# allow you to declare the variable you are using as to count iterations in the for loop:

For loops

- For loops are used to allow you to repeat sections of code a fixed, or variable amount of times. This allows you to make your code more compact and clean.
- There are several statements you can use to control a loop:

```
break;
```

- This allows you to exit a loop without finishing the loop.

```
for(int i = 0; i < 16; i++)
{
    if(i == 12)
    {
        break;
    }
}
```

- Exit the loop when i is 12, not when the loop would usually finish

For loops

```
    continue;
```

- This allows you to skip straight to the next iteration.

```
for(int i = 0; i < 16; i++)
{
    if(i % 2 == 1)
    {
        continue;
    }
    Console.WriteLine(i);
}
```

- Print only the even numbers by skipping the interactions where i is odd.

Switch Case

- In C#, Switch statement is a multiway branch statement. It provides an efficient way to transfer the execution to different parts of a code based on the value of the expression.

```
switch ( variable)
    {
        case1 :
        statement ;
        break;
        case..
    }
```

Switch Case

```
int nitem = 5;
      switch (nitem)
      {

      case 1:
          Console.WriteLine("case 1");
          break;

      case 5:
          Console.WriteLine("case 5");
          break;

      case 9:
          Console.WriteLine("case 9");
          break;

      default:
          Console.WriteLine("No match found");
          break;
      }
```

Array

- Arrays in C# are very similar to arrays in java. They are defined using the brackets operator [], and they are initialized using a list defined with curly braces. For example:

```
int[] nums = { 1, 2, 3, 4, 5 };
```

- We can also define an empty array like this:

```
int[] nums = new int[10];
```

- To get the size of the array, use the Length method.

```
int[] nums = new int[10];
Console.WriteLine(nums.Length);
```

- To access a specific item in the array, we use the brackets operator:

```
int[] nums = new int[10];
int firstNumber = nums[0];
int secondNumber = nums[1];
nums[2] = 10;
```

- Notice that C# uses zero based indices.

Multidimensional arrays

- C# supports multidimensional arrays, defined in the following manner:

```
int[,] matrix = new int[2,2];
matrix[0,0] = 1;
matrix[0,1] = 2;
matrix[1,0] = 3;
matrix[1,1] = 4;

int[,] predefinedMatrix = new int[2,2]
{ { 1, 2 }, { 3, 4 } };
```

List

- Lists in C# are very similar to lists in Java. A list is an object which holds variables in a specific order. The type of variable that the list can store is defined using the generic syntax. Here is an example of defining a list called numbers which holds integers.

```
List<int> numbers = new List<int>();
```

- The difference between a list and an array is that lists are dynamic sized, while arrays have a fixed size. When you do not know the amount of variables your array should hold, use a list instead.

List

- Once the list is initialized, we can begin inserting numbers into the list.

```
List<int> numbers = new List<int>();
numbers.Add(1);
numbers.Add(3);
```

- We can also add a whole array to a list using the AddRange function:

```
List<int> numbers = new List<int>();
int[] array = new int[] { 1, 2, 3 };
numbers.AddRange(array);
```

Removing from a list

- We can use Remove to remove an item from a list by specifying the item we want to remove.

```
List<string> fruits = new List<string>();

// add fruits

fruits.Add("apple");

fruits.Add("banana");

fruits.Add("orange");

// now remove the banana

fruits.Remove("banana"); Console.WriteLine(fruits.Count);
```

Removing from a list

- We can also use RemoveAt to specify an index of an item to remove. In our case, to remove the banana, we will use the index 1.

```
List<string> fruits = new List<string>();
// add fruits
fruits.Add("apple");
fruits.Add("banana");
fruits.Add("orange");
// now remove the banana
fruits.RemoveAt(1); Console.WriteLine(fruits.Count);
```

Concatenating lists

- We can use AddRange to join between lists.

```
List<string> food = new List<string>(); food.Add("apple");

food.Add("banana");

List<string> vegetables = new List<string>();

vegetables.Add("tomato"); vegetables.Add("carrot");

food.AddRange(vegetables); Console.WriteLine(food.Count);
```

Dictionaries

- Dictionaries are special lists, whereas every value in the list has a key which is also a variable. A good example for a dictionary is a phone book.

```
Dictionary<string, long> phonebook = new Dictionary<string, long>();
phonebook.Add("Alex", 4154346543); phonebook["Jessica"] = 4159484588;
```

- Notice that when defining a dictionary, we need to provide a generic definition with two types - the type of the key and the type of the value.
- In this case, the key is a string whereas the value is an integer.
- There are also two ways of adding a single value to the dictionary, either using the brackets operator or using the Add method.

Dictionaries

- To check whether a dictionary has a certain key in it, we can use the ContainsKey method:

```
Dictionary<string, long> phonebook = new Dictionary<string,
long>();

phonebook.Add("Alex", 415434543); phonebook["Jessica"] =
415984588;

if (phonebook.ContainsKey("Alex"))

{

   Console.WriteLine("Alex's number is " +   phonebook["Alex"]);

}
```

Remove an item from a dictionary

- To remove an item from a dictionary, we can use the Remove method. Removing an item from a dictionary by its key is fast and very efficient. When removing an item from a List using its value, the process is slow and inefficient, unlike the dictionary Remove function.

```
Dictionary<string, long> phonebook = new Dictionary<string, long>();

phonebook.Add("Alex", 415434543); phonebook["Jessica"] = 415984588; phonebook.Remove("Jessica");
Console.WriteLine(phonebook.Count);
```

HW1

- Write a program to add up from x to y.
- Write a program to have an array of type strings and then reverse the array element
- Write a program similar to your contact in your phone, name, and phone number. (use dictionary)
- By using while loop, write a program to have 2 arrays of type double (x and y) size of both 100, x values [-50,50] and based on y=(2x-1)/n calculate y and finally print both arrays