CMPE 1666
Intermediate Programming

Lecture 1- Review of C#

Some general notes

- For ICA’s and Labs, the deadlines will be specified on Brighspace. Penalties for these ICAs/labs will be as specified in the section “Marking of Assignments/Labs” on Brightspace
- As the ICA’s and Labs are released, you should ensure that you check the deadlines
- I’ll be giving time in class for working on ICAs and Labs. I expect that the time will be used for that purpose and that I’ll be able to see a number of commits on Github during these classes.
- Work time given for this class have to be used for activities of this class only
- I’m not going to checkoff the works of a student who only drops in the class for checkoff.
- Notifications about Lab Exam dates will be available on Brighspace, generally about 1-2 weeks prior to the lab exam. Ensure that you are aware of them.

C# Review

Selection statements: If-then-else v/s switch

- **Check the Testing Expression:** An if-then-else statement can test expressions based on ranges of values or conditions, whereas a switch statement tests expressions based only on a single integer, enumerated value, or String object.
- **Switch better for Multi way branching**
- **if-else better for boolean values:** If-else conditional branches are great for variable conditions that result into a boolean, whereas switch statements are great for fixed data values.
- **Speed:** A switch statement might prove to be faster than ifs provided number of cases are good. The compiler performs optimization.
- **Clarity in readability:** A switch looks much cleaner when you have to combine cases.

If-else

- E.g.

```
if (numValues >= 100){
      Console.WriteLine(“Value too big”);
      Console.WriteLine(“The value should be less than 100”);
      }
else{
     Console.WriteLine($“The input value was: {x}”);
      Console.WriteLine(“Value in proper range”);
     }
```

switch

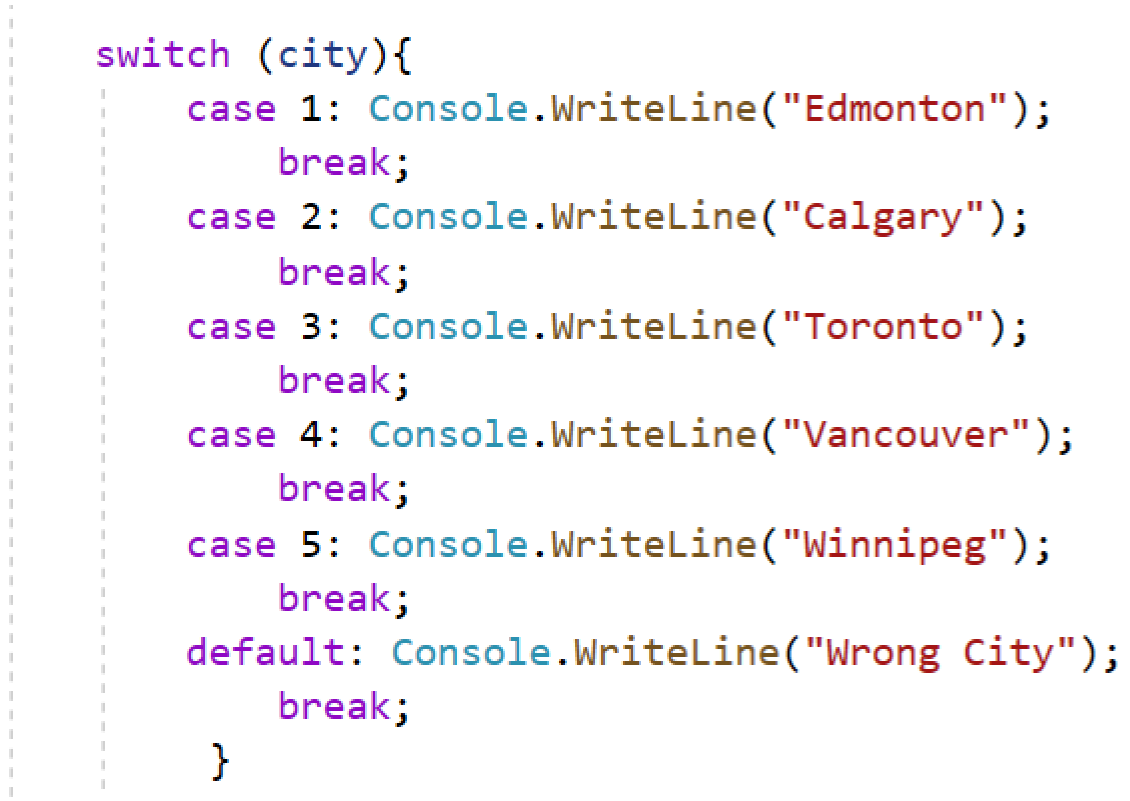
- Consider that we have the following **if..else** statements

```
if (city== 1)
    Console.WriteLine(“Edmonton”);
else if (city==2)
     Console.WriteLine(“Calgary”);
else if (city==3)
     Console.WriteLine(“Toronto”);
else if (city==4)
     Console.WriteLine(“Vancouver”);
else if (city==5)
      Console.WriteLine(“Winnipeg”);
else
     Console.WriteLine(“Wrong City”);
```

- Since each condition tests for the equality of a single value, we can use a **switch** statement, instead.

Switch

- The conditional block in the previous slide can be written as:



```
switch (city){
    case 1: Console.WriteLine("Edmonton");
        break;
    case 2: Console.WriteLine("Calgary");
        break;
    case 3: Console.WriteLine("Toronto");
        break;
    case 4: Console.WriteLine("Vancouver");
        break;
    case 5: Console.WriteLine("Winnipeg");
        break;
    default: Console.WriteLine("Wrong City");
        break;
 }
```

Lecture 1- Demo 1- Combining values in a switch

- In a switch block, cases can be combined.
- To illustrate this, we'll write a console program that accepts an in from a user for a month number, then display the Season based on the month number as below:
  - Months 12, 1, 2: Winter
  - Months 3,4,5: spring
  - Months 6,7,8 : summer
  - Months 9,10, 11: Fall
  - Other month numbers: error message

Loops

- Four Kinds of Loops in C#
  - while
  - do….while
  - for
  - Foreach
- All loops need to have some kind of control variable. Testing on this variable will determine whether we go to the next iteration or we leave the loop.
- The control variable can be a count or can refer to a value assigned through input or otherwise.
- Loops can be definite (executed a known number of times) or indefinite (exit depends on an input value)

while loops

- Most general- used to loop through a block of code 0 or more times (as long as a condition is true)

```
int i = 0;
while (i < 5)
{
    Console.Write(i + " ");
    i++;
}
```

- Can be used in all situations, but it’s preferred to be used when the number of iterations is not known beforehand. Number of iterations can also be 0.
- Note: we have initialization, update of control variable and condition testing

do..while loops

- Similar to while loops but execute at least once.
- Again, we note that we have initialization, condition testing and update of control variable

```
int i = 0;
do
{
    Console.Write(i + " ");
    i++;
} while (i < 5);
```

Note about while and do..while loops

- **while** and **do..while** loops are indefinite loops. So, they can be used when number of iterations is not known beforehand.
- E.g.

```
string ans="N"
while (ans !="Y"){
 ……………………………
 …………………………….
Console.Write("Do You want to Continue (Y/N)");
Console.ReadLine();
   }
```

- **while** and **do..while** loops are also useful for input validation.

for loops

- Used when loop will be executed a fixed number of times known beforehand.
- Initialization and update of the control variable as well as the condition testing is done within the loop header

```
for (int i = 0; i < 5; i++)
{
   Console.Write(i + "  ");
}
```

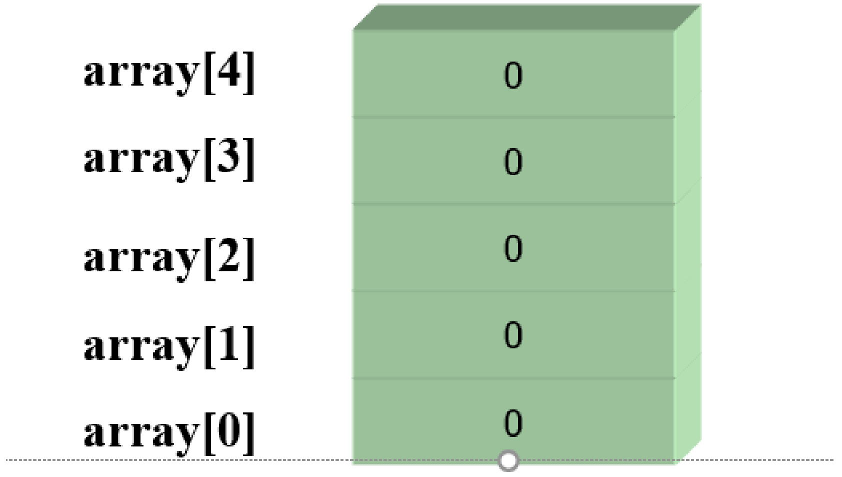
foreach loops

- **foreach** loops are used for iterating over arrays.

```
char[] gender = {'m','f','m','m','m','f','f','m','m','f'};
         int male = 0, female = 0;
         foreach (char g in gender)
         {
            if (g == 'm')
                  male++;
            else if (g =='f')
                  female++;
         }
```

Arrays

- Arrays are used when we want to store multiple elements.
- We can have arrays of basic types (int, char, double etc.) or arrays of any struct or object type.
- int[] array = new int[5];



Arrays

- Declaring an int array: **int[] myArray;**
- Creating the array to contain 10 elements: **myArray=new int[10];**
- Note that declaring and creating the array can both occur in the same statement:

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

Arrays

- We can also declare and initialize an array in the same statement

```
int[] myArray= new int[] {1, 2, 3, 4, 5}
```

- Accessing array elements is performed through using the array name and an index;
- E.g. **Console.WriteLine(myArray[2]);**

Iterating Through Arrays

- We can iterate through an array using **while**, **do..while** or **for** loops.
- However, use of a **for** loop is more common.
- E.g.

```
for (int i=0; i<myArray.Length; i++){
            myArray[i]=rand.Next(1,100);
      }
  for(int i=0; i<myArray.Length; i++)
              Console.WriteLine(myArray[i]);
```

Iterating Through Arrays

- Another common way of iterating through an array is through the use of the **foreach** loop.
- However, a foreach loop can only be used to access the elements of an array, not to modify them.
- E.g. **foreach(int x in myArray)**
- **Console.WriteLine(x);**

Lecture1-Demo1b

- To the program for Demo1, add the following:
  - Declare an array of strings called Month, containing the months of the year
  - Perform the display of all month numbers and corresponding month before the user enters a value.
  - After the user enters a value, display both the month name and the season.

Methods

- **Methods** are used to provide a modular programming approach.
- Methods can return values.
- They can have parameters.
- Parameters can be **value** type (default) or **reference** type.
- Reference type parameters are either **ref** or **out** – but we'll only use **out**
- **Methods** can also be Overloaded.

GDI Drawer

- GDI Drawer is a Shapes-drawing software developed at NAIT.
- The dll file and manual are available on GitHub.
- A copy has also been placed on Brightspace for your convenience.

Lecture1-Demo2- Review of GDIDrawer

Create a console application that performs the following:

- It creates a GDI Drawer window
- It executes a while (true) loop that continuously checks for a left-mouse click on the GDI Drawer window.
- When a left-mouse click occurs it creates a red circle of diameter 50 pixels with the center being the position of the click

Enums

- **Enums** are **user defined** types containing a number of specified values.
- **Windows** uses a number of enum types, which have already been defined. One of them is the **Color** type.
- Programmers can also define their own enum types

Eg: **public enum EDay {Working, Weekend, Holiday};**

- We can then declare and use variables of these types

E.g. **Eday eToday= Eday.Working**

Structs

- The struct is an aggregate type that allows us to combine data items of different types into one entity.
- E.g.

```
        private struct SGolfScoreType
        {
           public string _firstName;
           public string _lastName;
           public int _score;
        }

        static void Main(string[] args)
        {
           SGolfScoreType player1;
           player1._firstName = "Bob";
           player1._lastName = "Wert";
           player1._score = 78;

           Console.WriteLine($“Name: {player1._firstName} {p;layer1._lastName}, Score:
             {player1._score}”)
}
```

Declaring Structs and assigning member variables

- We can create a struct and assign values to each member as in the previous slide

```
SGolfScoreType player1;
 player1._firstName = "Bob";
 player1._lastName = "Wert";
 player1._score = 78;
```

- Or we can use a struct initializer

```
 SGolfScoreType player1= new SGolfScoreType{_firstName="Bob", _lastName="Wert",
_score= 78};
```

Lecture1-Demo2b

- To the program of Demo2, add a struct called **circle** containing, as members, a center (a point), a diameter and a color.
- To the Main method, add an array of size 50 of the above struct type.
- Modify the code in Main() such that on a left click, a ball is created with a random color and a random diameter in the range of 25 to 75. The ball is assigned to the next available element of the array and also displayed on the GDIDrawer window. Keep a count of how many balls have been added to the array.
- To the program add a method Render() that has as parameters a GDIDrawer window, an array of Circle structs, and a count. The count represents the number of structs assigned to the array. The program clears the GDI Drawer window, iterates over the array and for all assigned structs and redraws them.
- To Main(), add the code such that if there is a right-click, the program iterates over the array and for each circle with diameter at size 50 or higher, it changes the color to blue. The program then calls Render() to redraw all the circles.

Value Type - variables

- Consider

```
int x= 7;
int y=x;
   x=10;
```

What will **Console.WriteLine($”x:{x}, y:{y}”);** display?

Reference Type variables

- Now consider:

```
  int[] a=new int[1];
   a[0]= 17;
   int[] b;
    b=a;
    a[0]= 25;
```

What will **Console.WriteLine($“a[0]:{a[0]},b[0]:{b[0]}”);** give us?

Structs v/s arrays

- A struct is a value type as opposed to an array which is a reference type.
- Hence, if we have 2 struct variables of type **SGolfScoreType** as below, they will be 2 different objects in memory

```
SGolfScoreType player1= new SGolfScoreType{_firstName="Bob", _lastName="Wert", _score= 78};

player2=player1;
```

- If we have 2 array variables as below, they will refer to the same object

```
int[] array1={20,30,40,50,70};

int[] array2= array1;
```

Precedence

- Order of precedence matters when performing operations.
- E.g. We know that the result of 6 + 7 * 8 is 62 and not 104
- But Consider that A is True, B is True and C is False. What is the value of A or B and C ?
- Let x be 7.25- What’s the difference between
- (int) x*100 and (int)(x*100)

Associativity

- For binary operators of the same order of precedence, associativity is generally left to right.
- E.g. consider R= (b*b -4a*c)/2*a

- However, for the assignment operator, associativity is right to left.
- x=y=z=4;