Programming in C#
Language Overview

Overview
- Quickly go through some of the major differences between C++ or Java and C#.
- Overview of a C# program.
- Built-in C# types.
- Canonical Hello World program
- Array

Language
- Namespaces
- Classes
- Fields
- Properties
- Methods
- Attributes
- Events
- Interfaces (contracts)
- Methods
- Properties
- Events
- Control Statements
- if, else, while, for, switch
- foreach
- Additional Features
- Operation Overloading
- Structs
- Enums
- Delegates
- OO Features
- Type Unification
- Inheritance
- Polymorphism

C# Program Structure
- Namespaces
- Contain types and other namespaces
- Type declarations
- Classes, struct’s, interfaces, enum’s, and delegates
- Contain members
- Members
- Constants, fields, methods, operators, constructors, destructors
- Properties, indexers, events
- Organization
- No header files, types imported using assemblies (dll’s)
- Type reflection

Major Language Differences
- Automatic memory management
  - Garbage Collection
  - No pointers
- Everything inherits from System.Object
- Additional language constructs to aid in implementing common patterns:
  - Iterators – foreach and yield statements
  - Data encapsulation – Properties
  - Function pointers or Functors – delegates
  - Observer Design Pattern – events
  - Aspect-oriented programming – Attributes (loosely)
  - Functional programming – LINQ and anonymous methods

Other Language Differences
- Salient points
  - Conditionals must evaluate to a Boolean
  - No pointers. Use the dot ".” to access both namespaces and fields/methods.
  - All fields are initialized by the CLR (zero for value types, null for reference types).
  - The switch statement does not “fall-thru” (unless empty)
  - The switch statement can take bool’s, enum’s, integral types and strings.
  - Expressions must be useful (no a==b:).
  - Internal or assembly protection control.
C# 5.0 Features

- Built-in Types
  - C# predefined types
    - The "root" object
    - Logical bool
    - Signed sbyte, short, int, long
    - Unsigned byte, ushort, uint, ulong
    - Floating-point float, double, decimal
    - Textual char, string
      - Textual types use Unicode (16-bit characters)

Common Value Types

- All integral types: int, uint, long, short, byte, ...
- float’s, double’s and decimal’s
- Structs
  - Aka user-defined value (light-weight) types
  - Can contain arbitrary data
  - Non-extensible (sealed subclasses)
  - System.Decimal is a struct.

Instance Initialization

- Must be initialized (unlike C++)
  - double x; // x == 0
  - string f; // f.equals("")
  - A a; // a == null
- Fundamental difference between double and class A?
  - Reference types vs. value types

Built-in Types - Strings

- Strings are first-class immutable objects in C#
- Behave like a value type
- Can be indexed like a char array.
- Has many methods (see System.String)
- The @ command allows specifying a string literal spanning lines.
  ```csharp
  string s = "Hello";
  char third = s[2];
  string[] split = s.Split(third);
  ```

Reference Types

- Classes, including user-defined classes
  - Inherited from System.Object
  - Transparently refers to a memory location
  - Similar to pointers in other languages
  - Other view is that: Everything is a pointer.
  - Can be set to null

```csharp
A a = new A();
A b = a;
```
Value Types
- Contain the actual value, not the location
- Inherited from System.ValueType
  - Which is (logically) derived from System.Object.
- Treated specially by the runtime
- Need to be boxed in order to get a reference

```csharp
int a = 137;
int b = a;
```

Boxing and Unboxing
- Value types are not indirectly referenced
  - More efficient.
  - Less memory.
- In order to treat all types uniformly, a value type needs is converted to a reference type.
  - called boxing. Reverse is unboxing

```csharp
{ int a = 137;
  object o1 = a;
  object o2 = o1; // Boxing
  int b = (int)o2; //Unboxing
}
```

Differences Between Types
- Copy semantics:
  - Polynomial a = new Polynomial();
  - Polynomial b = a;
  - b.Coefficient[0] = 10;
  - Console.WriteLine(a.Coefficient[0]);
  - int a = 1;
  - int b = a;
  - b = 10;
  - Console.WriteLine(a);
- Copies of value types make a real copy
  - Important for parameter passing
  - Boxing still copies

Function Parameters
- In C++ you could pass a variable by:
  - Value (creating a new copy and passing it)
  - Pointer (passing in the address)
  - Reference (same as pointer, less cluttered syntax).
- In C# there seems to be some confusion and misinformation (including C# in a Nutshell) on this.
  - The MSDN documentation for ref has this correct.
- If you were raised on pointers and understand this (which you are not), then it is quite simple.
  - The ref and out keywords indicate that you should pass:
    - Value types — address of value type is passed.
    - Reference types — address of pointer is passed.

Function Parameters
- ref parameters
  - Use the value that is passed in
  - Change the value of a value type passed in
  - Change the instance that a reference type points to.
  - A valid (or initialized) instance or value type must be passed in.
  - Null is a valid value.

Function Parameters
- out parameters
  - Set the value of a value type passed in.
  - Set the instance that a reference type points to.
  - Does not need to be initialized before calling.
  - Compiler forces you to set a value for all possible code paths.
Function Parameters

- For variable number of parameters
  
  ```
  public void f(int x, params char[] ar);
  ```

  ```
  call f(1), f(1, 's'), f(1, 's', 't'), f(1, 'sf'.ToCharArray());
  ```

- Must be the last argument in the list

Control Statements

- If, else

```java
if (true) {
  // Some code
}
else {
  // Some other code
}
```

- while

```java
while (true) {
  // Some code
}
```

- for

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

- foreach

```java
foreach (int i in list) {
  // Some code
}
```

For Loops

- The typical **for** loop syntax is:

  ```java
  for (initialization; test; update) {
    statements;
  }
  ```

  - Consists of
    - Initialization statement
    - Boolean test expression
    - Update statement
    - Loop body block

- foreach Loop

  **Iteration over a Collection**

  ```java
  foreach (Type element in collection) {
    statements;
  }
  ```

  - Iterates over all elements of a collection
    - The **element** is the loop variable that takes sequentially all collection values
    - The **collection** can be list, array or other group of elements of the same type
foreach Loop – Example

- Example of `foreach` loop:

```csharp
string[] days = new string[] {
    "Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday", "Sunday"};
foreach (String day in days)
{
    Console.WriteLine(day);
}
```

- The above loop iterates of the array of days
- The variable day takes all its values

Processing Arrays Using `foreach`

- Print all elements of a `string[]` array:

```csharp
string[] capitals = {
    "Ankara",
    "Washington",
    "London",
    "Paris"
};
foreach (string capital in capitals)
{
    Console.WriteLine(capital);
}
```

foreach Loop

Live Demo example 05; 06 and 07

Switch Statement

- Switch statements must be expressions that can be statically evaluated.
- Restricted to primitive types, strings and enums.
- There is no fall through on switch cases unless the case is an empty case:

```csharp
switch (myEmployee.Name)
{
    case "Boss":
        Console.WriteLine("Your fired!");
        break;
    case "Henry":
    case "Jane":
        Console.WriteLine("I need a pay raise.");
        break;
    default:
        Console.WriteLine("Busy working.");
        break;
}
```

Jump Statements

- Several C# statements provide a break in the execution:
  - `break` – jumps out of a while or for loop or a switch statement.
  - `continue` – starts the next iteration of a loop.
  - `goto` – do not use.
  - `return` – returns out of the current method.
  - `throw` – Causes an exception and ends the current try block or method recursively.
Declaring and Creating Arrays

What are Arrays?
- An array is a sequence of elements
- All elements are of the same type
- The order of the elements is fixed
- Has fixed size (Array.Length)

Declaring Arrays
- Declaration defines the type of the elements
- Square brackets [ ] mean "array"
- Examples:
  - Declaring array of integers:
    ```java
    int[] myIntArray;
    ```
  - Declaring array of strings:
    ```java
    string[] myStringArray;
    ```

Array Declarations
- An array is an object (not just a stream of objects). See System.Array.
- Bounds checking is performed for all access attempts.
- Declaration similar to Java, but more strict.
- Type definition: a is a “1D array of int’s”
- Instance specifics: a is equal to a 1D array of int’s of size 10.
```
    int[] a = new int[10];
    int[] b = a;
    int[] c = new int[3] {1,2,3};
    int[] d = {1,2,3,4};
```

Creating Arrays
- Use the operator
  - Specify array length
- Example creating (allocating) array of 5 integers:
  ```java
  myIntArray = new int[5];
  ```

Creating and Initializing Arrays
- Creating and initializing can be done together:
```java
    myIntArray = {1, 2, 3, 4, 5};
```
- The new operator is not required when using curly brackets initialization
Creating Array – Example

- Creating an array that contains the names of the days of the week

```java
String[] daysOfWeek = {
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday"
};
```

Days of Week

How to Access Array Element?

- Array elements are accessed using the square brackets operator `[]` (indexer)
  - Array indexer takes element's index as parameter
  - The first element has index 0
  - The last element has index `Length - 1`
- Array elements can be retrieved and changed by the `[]` operator

Accessing Array Elements

Read and Modify Elements by Index

Reversing an Array – Example

- Reversing the contents of an array

```java
int[] array = new int[] {1, 2, 3, 4, 5};
// Get array size
int length = array.Length;
// Declare and create the reversed array
int[] reversed = new int[length];
// Initialize the reversed array
for (int index = 0; index < length; index++)
{
    reversed[length-index-1] = array[index];
}
```
Reading Arrays From the Console

- First, read from the console the length of the array
  ```csharp
  int n = int.Parse(Console.ReadLine());
  ```
- Next, create the array of given size and read its elements in a `for` loop
  ```csharp
  int[] arr = new int[n];
  for (int i = 0; i < n; i++)
  {
    arr[i] = int.Parse(Console.ReadLine());
  }
  ```

Symmetry Check – Example

- Read `int` array from the console and check if it is symmetric:
  ```csharp
  bool isSymmetric = true;
  for (int i = 0; i < (array.Length + 1) / 2; i++)
  {
    if (array[i] != array[n - i - 1])
    {
      isSymmetric = false;
    }
  }
  ```

Printing Arrays on the Console

- Process all elements of the array
- Print each element to the console
- Separate elements with white space or a new line
  ```csharp
  string[] array = {"one", "two", "three"};
  // Process all elements of the array
  for (int index = 0; index < array.Length; index++)
  {
    // Print each element on a separate line
    Console.WriteLine("element[{0}] = {1}", index, array[index]);
  }
  ```

Array Example- initialize & write

```csharp
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
      int[,] n = new int[10];  // n is an array of 10 integers
      int i, j;
      for (i = 0; i < 10; i++)
      {
        n[i] = i + 100;
      }
      for (j = 0; j < 10; j++)
      {
        Console.WriteLine("Element[{0}] = {1}", j, n[j]);
      }
    }
  }
}
```

Accessing The Elements of Multidimensional Arrays

- Accessing N-dimensional array element:
  ```csharp
  nDimensionalArray[index1, ..., indexn]
  ```
- Getting element value example:
  ```csharp
  int[,] array = { (1, 2), (3, 4) }
  int element11 = array[1, 1];  // element11 = 4
  ```
- Setting element value example:
  ```csharp
  int[,] array = new int[3, 4];
  for (int row=0; row < array.GetLength(0); row++)
  {
    for (int col=0; col < array.GetLength(1); col++)
    {
      array[row, col] = row + col;
    }
  }
  ```

MultiArray Example

```csharp
using System;
namespace ArrayApplication
{
  class MyArray
  {
    static void Main(string[] args)
    {
      int[,] n = new int[10, 20];
      int[,] n = new int[10, 20];
      for (int row=0; row<10; row++)
      {
        for (int col=0; col<20; col++)
        {
          n[row, col] = row + col;
        }
      }
    }
  }
}
```
Reading Matrix – Example

- Reading a matrix from the console
  ```csharp
  int rows = int.Parse(Console.ReadLine());
  int columns = int.Parse(Console.ReadLine());
  int[,] matrix = new int[rows, columns];
  string inputNumber;
  for (int row = 0; row < rows; row++)
  {
    for (int column = 0; column < columns; column++)
    {
      Console.Write("matrix[\{0\},\{1\}] = ", row, column);
      inputNumber = Console.ReadLine();
      matrix[row, column] = int.Parse(inputNumber);
    }
  }
  ```

- Printing a matrix on the console:
  ```csharp
  for (int row = 0; row < matrix.GetLength(0); row++)
  {
    for (int col = 0; col < matrix.GetLength(1); col++)
    {
      Console.Write("(\{0\},\{1\}) = ", row, col);
      Console.WriteLine(matrix[row, col]);
    }
    Console.WriteLine();
  }
  ```

Jagged Arrays

- Can have standard C-style `jagged arrays`
  ```csharp
  int[] array = new int[30];
  int[][] array = new int[2][];
  array[0] = new int[100];
  array[1] = new int[1];
  ```
- Stored in random parts of the heap
- Stored in row major order

C# Arrays

- Multi-dimensional arrays are jagged arrays with a `user-enforced` constraint in C.
- Really just 1D arrays, but each element can contain a 1D array (recursion).
- C# provides `true` multi-dimensional arrays
- Elements are stored sequentially
- CLR (JIT compiler) computes the offset code
  ```csharp
  int[,] array = new int[10,30];
  array[3,7] = 137;
  ```
- CLR (JIT compiler) computes the offset code
  ```csharp
  int[,] arr4 = new int[2,3] { {1,2,3}, {4,5,6} };
  int[,] arr5 = new int[1] { {1,2,3}, {4,5,6} };
  int[,] arr6 = { {1,2,3}, {4,5,6} };
  ```

Indexers

- Allow bracket notation on any object
  ```csharp
  public string this[int a, double b] {
    ... }
  ```
- Related to C++ operator[ ] overloading
- Special property

Indexers

- Lets an instance behave as a virtual array
- Can be overloaded
- Can be read-only, write-only, or read/write

```csharp
public class ListBox : Control {
  public string this[int index] {
    get { return items[index]; }
    set { items[index] = value; Repaint(); }
  }
}
```

```csharp
ListBox listBox = new ListBox();
lListbox[0] = "hello";
Console.WriteLine(listBox[0]);
```
Lists

- **List**s are arrays that resize dynamically
  - When adding or removing elements
  - Also have indexers (like `Array`)
  - `T` is the type that the List will hold
    - E.g. `List<int>` will hold integers
    - `List<object>` will hold objects
- Basic Methods and Properties
  - `Add(T element)` – adds new element to the end
  - `Remove(element)` – removes the element
  - `Count` – returns the current size of the List

**List Example**

```csharp
List<int> intList = new List<int>();
for (int i = 0; i < 5; i++)
{
    intList.Add(i);
}
```

**List vs. Arrays**

- Lets have an array with capacity of 5 elements
  ```csharp
  int[] intArray = new int[5];
  ```
- If we want to add a sixth element (we have already added 5) we have to do
  ```csharp
  int[] copyArray = intArray;
  for (int i = 0; i < 5; i++)
  {
      intArray[i] = copyArray[i];
  }
  ```
- With List we simply do
  ```csharp
  list.Add(newValue);
  ```

**Copying Arrays**

- Sometimes we must copy the values from one array to another one
  - If we do it the intuitive way
    ```csharp
    // Copy the source to the target. //
    Array.Copy(source, target, n);
    ```
    - `n` size of the array

**Summary**

- Arrays are a fixed-length sequences of elements of the same type
- Array elements are accessible by index
  - Can be read and modified
- Iteration over array elements can be done with `for` and `foreach` loops
- Matrices (2-dimensional arrays) are very useful for presenting tabular data
Write a program that allocates array of 20 integers and initializes each element by its index multiplied by 5. Print the obtained array on the console.

2. Write a program that reads two arrays from the console and compares them element by element.

3. Write a program that compares two char arrays lexicographically (letter by letter).

4. Write a program that finds the maximal sequence of equal elements in an array.

Example: \{2, 1, 2, 3, 3, 2, 2, 1\} \rightarrow \{2, 2, 2\}.

5. Write a program that finds the maximal increasing sequence in an array. Example:
\{3, 2, 3, 4, 2, 2, 4\} \rightarrow \{2, 3, 4\}.

6. Write a program that reads two integer numbers N and K and an array of N elements from the console. Find in the array those K elements that have maximal sum.

7. Sorting an array means to arrange its elements in increasing order. Write a program to sort an array. Use the "selection sort" algorithm: Find the smallest element, move it at the first position, find the smallest from the rest, move it at the second position, etc.

8. Write a program that finds the sequence of maximal sum in given array. Example:
\{2, 3, -6, -1, 2, -1, 6, 4, -8, 8\} \rightarrow \{2, -1, 6, 4\}

Can you do it with only one loop (with single scan through the elements of the array)?

9. Write a program that finds the most frequent number in an array. Example:
\{4, 1, 1, 4, 2, 3, 4, 1, 2, 4, 9, 3\} \rightarrow 4 \text{ (5 times)}

10. Write a program that finds in given array of integers a sequence of given sum S (if present). Example:
\{4, 3, 1, 4, 2 \leq 8\}, S=11 \rightarrow \{4, 2, 5\}

11. Write a program that fills and prints a matrix of size \(n \times n\) as shown below: (examples for \(n = 4\))

\[\begin{array}{cccc}
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15 \\
4 & 8 & 12 & 16 \\
\end{array}\]

12. Write a program that reads a rectangular matrix of size \(N \times M\) and finds in it the square \(3 \times 3\) that has maximal sum of its elements.

13. We are given a matrix of strings of size \(N \times M\). Sequences in the matrix we define as sets of several neighbor elements located on the same line, column or diagonal. Write a program that finds the longest sequence of equal strings in the matrix. Examples:

\[\text{ha, ha, ha} \quad \text{and} \quad \text{s, s, s, s} \]
Exercises (6)

14. Write a program that creates an array containing all letters from the alphabet (A-Z). Read a word from the console and print the index of each of its letters in the array.

15. Write a program that finds the index of given element in a sorted array of integers by using the binary search algorithm (find it in Wikipedia).

16. Write a program that sorts an array of integers using the merge sort algorithm (find it in Wikipedia).

17. Write a program that sorts an array of strings using the quick sort algorithm (find it in Wikipedia).

Exercises (7)

18. Write a program that finds all prime numbers in the range [1...10 000 000]. Use the sieve of Eratosthenes algorithm (find it in Wikipedia).

19. * We are given an array of integers and a number S. Write a program to find if there exists a subset of the elements of the array that has a sum S. Example: arr=[2, 1, 2, 4, 3, 5, 2, 6], S=14 → yes (1+2+5+6)

20. * Write a program that reads three integer numbers N, K and S and an array of N elements from the console. Find in the array a subset of K elements that have sum S or indicate about its absence.

Exercises (8)

21. * Write a program that reads an array of integers and removes from it a minimal number of elements in such way that the remaining array is sorted in increasing order. Print the remaining sorted array. Example: {6, 1, 4, 3, 0, 3, 6, 4, 5} → {1, 3, 3, 4, 5}

22. * Write a program that reads a number N and generates and prints all the permutations of the numbers [1 ... N]. Example:

n = 3 → {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}

Exercises (9)

23. Write a program that reads two numbers N and K and generates all the variations of K elements from the set [1..N]. Example:

N = 3, K = 2 → {1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {2, 3}, {3, 1}, {3, 2}, {3, 3}

24. Write a program that reads two numbers N and K and generates all the combinations of K distinct elements from the set [1..N]. Example:

N = 5, K = 2 → {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {3, 5}, {4, 5}

Exercises (10)

25. Write a program that fills a matrix of size (N, N) as shown in the examples (for N=4):

| 16 15 13 10 | 14 12 9 6 |
| 11 8 5 3 | 7 4 2 1 |
| 7 11 14 16 | 3 6 10 |

References

- Prof. Roger Crawfis
  - http://www.cse.ohio-state.edu/~crawfis/cse459_CSharp/index.html
  - Pls listen the podcast about the chapter 2
  - These slides are changed and modified

- Programming C#, 4th Edition - Jesse Liberty – Chapter 2