

# **Introduction to Programming**

Lesson 1

# Objectives

---

Skills/Concepts	MTA Exam Objectives
Understanding Computer Programming	Understand computer storage and data types (1.1)
Understanding Decision Structures	Understand computer decision structures (1.2)
Understanding Repetition Structures	Identify the appropriate method for handling repetition (1.3)
Understanding Exception Handling	Understand error handling (1.4)

# Algorithms






---

- Algorithm refers to a method for solving problems.
- Common techniques for representing an algorithms:

Flowcharts and Decision Tables	
More precise than natural languages	Less formal and easier to use than programming languages

# Flowcharts

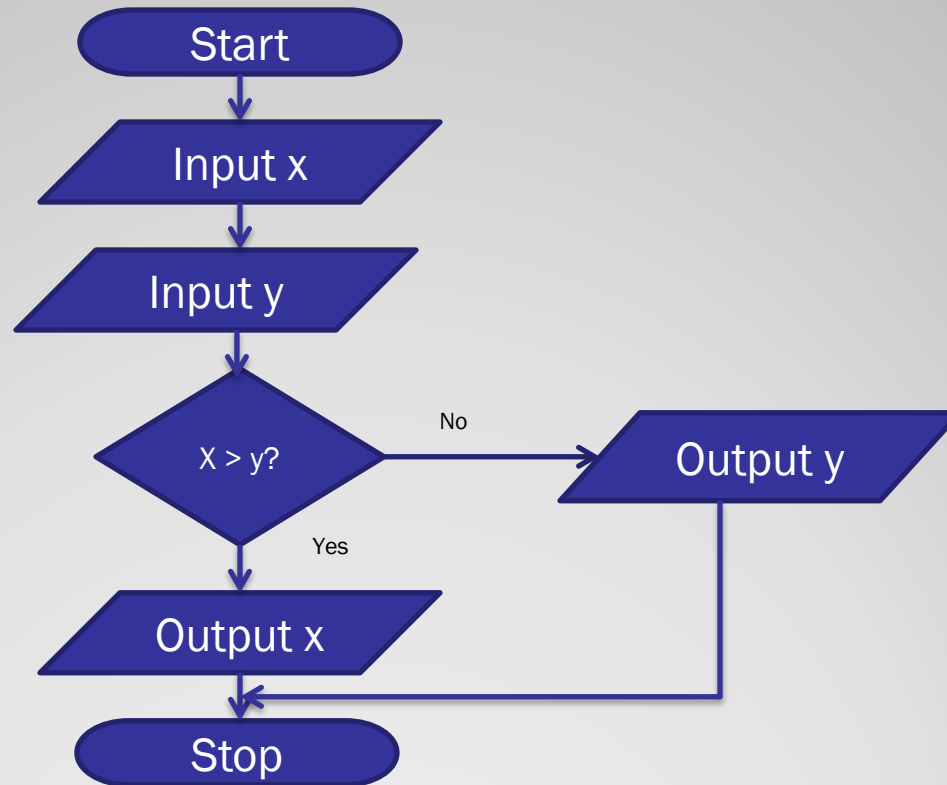
- A flowchart is a graphical representation of an algorithm.

Common Flowchart Symbols	
	Start or end of an algorithm
	Process or computational operation
	Input or out operation
	Decision making operation
	Direction of the flow of control

# Flowchart Example

---

- A flowchart that compares two numbers:



## Decision Table

---

- Useful for large number of conditions
- Compact and readable format
- A decision table to calculating discount:

Quantity < 10	Y	N	N	N
Quantity < 50	Y	Y	N	N
Quantity < 100	Y	Y	Y	N
Discount	5%	10%	15%	20%

# Introducing C#

---

- Microsoft .NET Framework
  - An Execution Environment
  - Reusable Class Libraries
  - Language Compilers
- The C# Programming Language
  - Part of the .NET Framework
  - High-level Language
  - Program needs to be compiled before they can be executed.
  - Case sensitive

# Structure of a C# Program

---

```
1  using System;
2  namespace Lesson01
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              Console.WriteLine("hello, world!");
9          }
10     }
11 }
```



# Elements of a C# Program

---

- Select common elements of a C# program:

<b>Data Types</b>	Types of data in a program. Common data types are int (integers), char (single character value), float (floating point values).
<b>Variables</b>	Provides temporary storage during program execution. <code>int number = 10;</code>
<b>Constants</b>	Data fields whose value cannot be modified. <code>const int i = 10;</code>
<b>Arrays</b>	A collection of items in which each item can be accessed by a unique index. <code>int[] numbers = { 1, 2, 3, 4, 5 };</code>
<b>Operators</b>	Symbols that specify which operation to perform on operands before returning a result.
<b>Methods</b>	Methods are code blocks containing a series of statements. Methods can receive input via arguments and can return a value to the caller.

# Decision Structures

---

The if  
Statement

The if-else  
Statement

The  
switch  
Statement

# The if Statement

---

- The **if** statement will execute a given sequence of statements only if the corresponding Boolean expression evaluates to true.

```
int number1 = 10;  
int number2 = 20;  
if (number2 > number1)  
{  
    Console.WriteLine("number2 is greater than number1");  
}
```

# The if-else Statement

---

- The **if-else** statement allows your program to perform one action if the Boolean expression evaluates to true and a different action if the Boolean expression evaluates to false.

```
public static void TestIfElse(int n)
{
    if (n < 10)
    {
        Console.WriteLine("n is less than 10");
    }
    else if (n < 20)
    {
        Console.WriteLine("n is less than 20");
    }
    else
    {
        Console.WriteLine("n is greater than or equal to 20");
    }
}
```

# The switch Statement

---

- The **switch** statement allows multi-way branching. In many cases, using a switch statement can simplify a complex combination of if-else statements.

```
public static void TestSwitch(int op1, int op2, char opr)
{
    int result;
    switch (opr)
    {
        case '+':
            result = op1 + op2;
            break;
        case '-':
            result = op1 - op2;
            break;
        default:
            Console.WriteLine("Unknown Operator");
            return;
    }
    Console.WriteLine("Result: {0}", result);
    return;
}
```

# **Repetition Structures**

---

The while Loop

The do-while Loop

The for Loop

The foreach Loop

Recursion

## The while Loop

---

- The **while** loop repeatedly executes a block of statements until a specified Boolean expression evaluates to false.

```
int i = 1;
while (i <= 5)
{
    Console.WriteLine("The value of i = {0}", i);
    i++;
}
```

## The do-while Loop

---

- The **do-while** loop repeatedly executes a block of statements until a specified Boolean expression evaluates to false. The **do-while** loop tests the condition at the bottom of the loop.

```
int i = 1;
do
{
    Console.WriteLine("The value of i = {0}", i);
    i++;
}
while (i <= 5);
```



# The for Loop

---

- The **for** loop combines the three elements of iteration—the initialization expression, the termination condition expression, and the counting expression—into a more readable code.

```
for (int i = 1; i <= 5; i++)  
{  
    Console.WriteLine("The value of i = {0}", i);  
}
```

## The foreach Loop

---

- The **foreach** loop is an enhanced version of the for loop for iterating through collections such as arrays and lists.

```
int[] numbers = { 1, 2, 3, 4, 5 };  
foreach (int i in numbers)  
{  
    Console.WriteLine("The value of i = {0}", i);  
}
```

# Recursion

---

- Recursion is a programming technique that causes a method to call itself in order to compute a result.

```
public static int Factorial(int n)
{
    if (n == 0)
    {
        return 1; //base case
    }
    else
    {
        return n * Factorial(n - 1); //recursive case
    }
}
```

# Exception Handling

---

- An exception is an unexpected error condition that occurs during program execution.
- When exception occurs, the runtime creates an exception object and “throws” it.
- Unless you “catch” the exception, the program execution will terminate.
- Exceptions are an object of the System.Exception class or one of its derived classes.
  - Example: **DivideByZeroException** exception object is thrown when the program attempts to divide by zero.
  - Example: **FileNotFoundException** exception object is thrown when the program cannot find a given file.

# Unhandled Exceptions

---

- What happens when the file c:\data.txt is not found in this code?

```
private static void ExceptionTest()
{
    StreamReader sr = null;
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
```

## Handling Exceptions with try-catch

---

- Place the code that throws the exceptions inside a **try** block.
- Place the code that handles an exception inside a **catch** block.
- You can have more than one catch blocks for each try block. Each catch block handles a specific exception type.
- A try block must have at least a catch block or a finally block associated with it.

# Exception Handling Sample

---

```
private static void ExceptionTest()
{
    StreamReader sr = null;
    try
    {
        sr = File.OpenText(@"c:\data.txt");
        Console.WriteLine(sr.ReadToEnd());
    }
    catch (FileNotFoundException fnfe)
    {
        Console.WriteLine(fnfe.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

# The finally Block

---

- The **finally** block is used in association with the try block.
- The **finally** block is always executed regardless of whether an exception is thrown.
- The **finally** block is often used to write clean-up code.

```
StreamReader sr = null;
try
{
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
finally
{
    if (sr != null)
    {
        sr.Close();
    }
}
```



# try-catch-finally Example

---

```
StreamReader sr = null;
try
{
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
catch (FileNotFoundException fnfe)
{
    Console.WriteLine(fnfe.Message);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    if (sr != null)
    {
        sr.Close();
    }
}
```

# Recap

---

- Algorithms
  - Flowchart, decision table
- C# Programming Language
  - Variables, constants, data types, arrays, operators, methods
- Decision Structures
  - if, if-else, switch
- Repetition Structures
  - while, do-while, for, foreach, recursion
- Exception Handling
  - try-catch, try-finally, try-catch