SIT771 – Lecture 3

Control flow, error detection, and error handling

Further reading

- Paul Deitel and Harvey Deitel (2018). Visual C# how to Program (6th ed). Pearson. Ebook on Deakin Library – Chapter 4, Chapter 5, Chapter 6, and Chapter 13.

Outline

In this lecture...

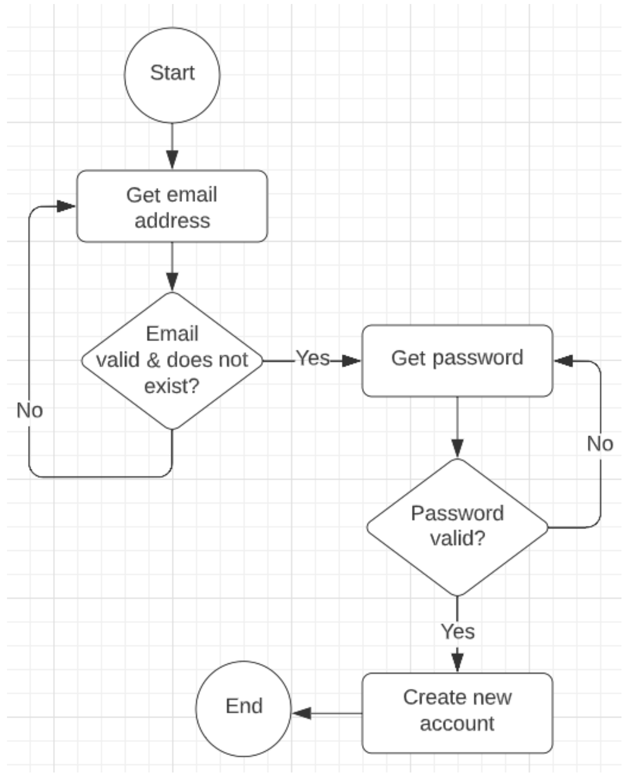
- Control flow
  - Sample scenario
  - Programming sense
  - Conditional statements
  - Loops
- Errors and error detection
  - The effect of errors
  - Error detection scenarios
- Error handling
  - Using specific methods
  - Try-catch exception handling
  - .Net exception classes

CONTROL FLOW

Sample scenario

Account registration workflow

- Step 1. User provides an email address.
- Step 2. System checks if the address is valid and does not already exist.
    - Step 2.1. If so, go to step 3.
    - Step 2.2. Otherwise, go to step 1.
- Step 3. User provides a password.
- Step 4. System checks if the password is valid.
    - Step 4.1. If it is, go to step 5.
    - Step 4.2. Otherwise, go to step 3.
- Step 5. System creates a new account.
- Step 6. Process completes.

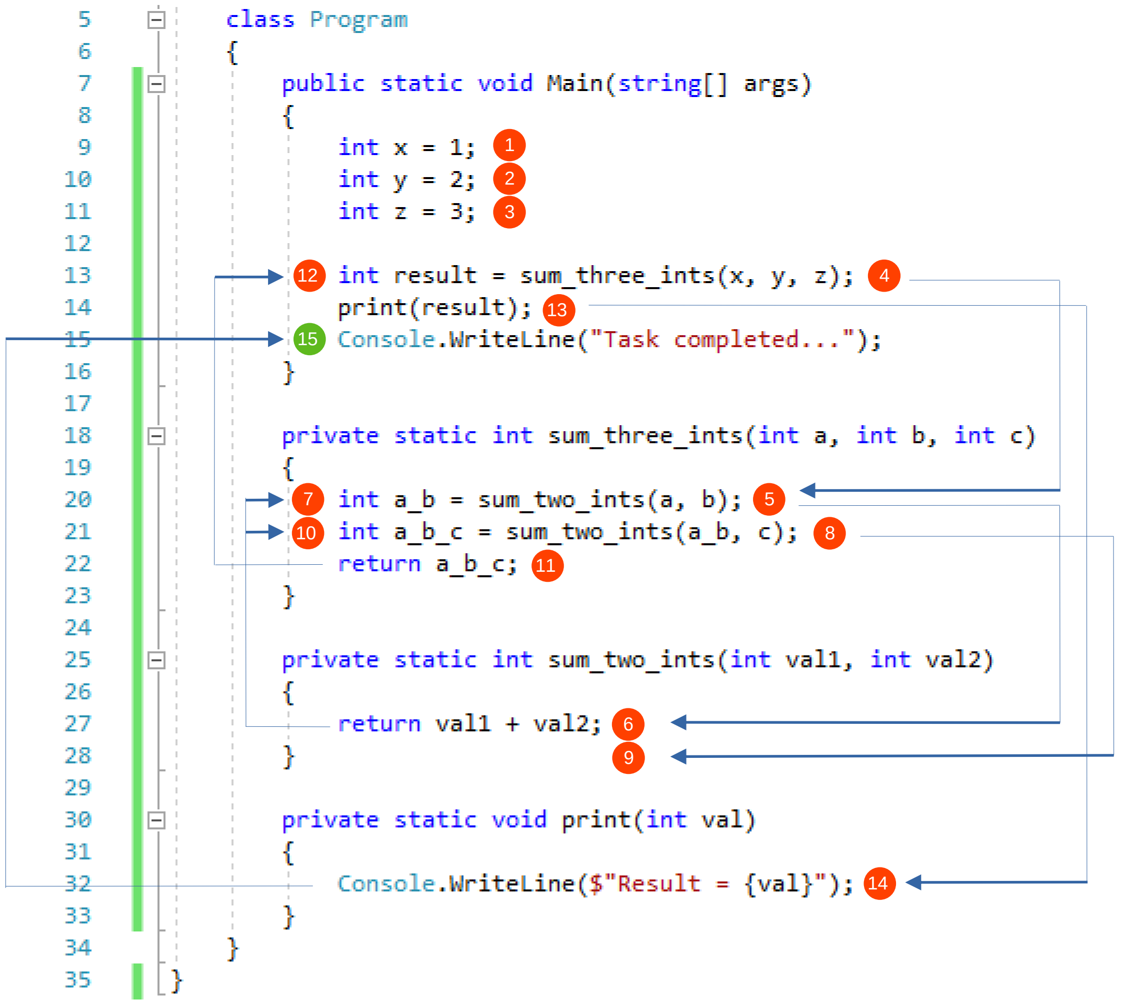


Flowchart created in LucidChart.

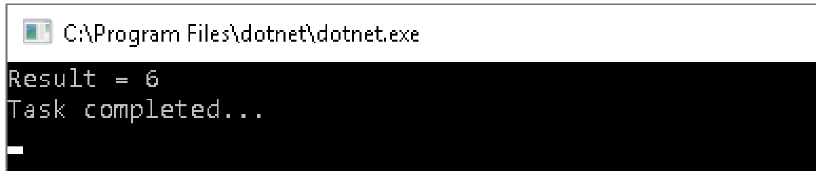
Programming sense

In computer science/codes...

- Control flow is...
  - The order in which individual statements, instructions, or function calls of an imperative program are executed or evaluated. [Wikipedia]
  - Should note...
    - If statements
    - Switch-case statements (or goto)
    - Method calls
    - Exceptions



```
class Program
{
    public static void Main(string[] args)
    {
        int x = 1;
        int y = 2;
        int z = 3;

        int result = sum_three_ints(x, y, z);
        print(result);
        Console.WriteLine("Task completed...");
    }

    private static int sum_three_ints(int a, int b, int c)
    {
        int a_b = sum_two_ints(a, b);
        int a_b_c = sum_two_ints(a_b, c);
        return a_b_c;
    }

    private static int sum_two_ints(int val1, int val2)
    {
        return val1 + val2;
    }

    private static void print(int val)
    {
        Console.WriteLine($"Result = {val}");
    }
}
```



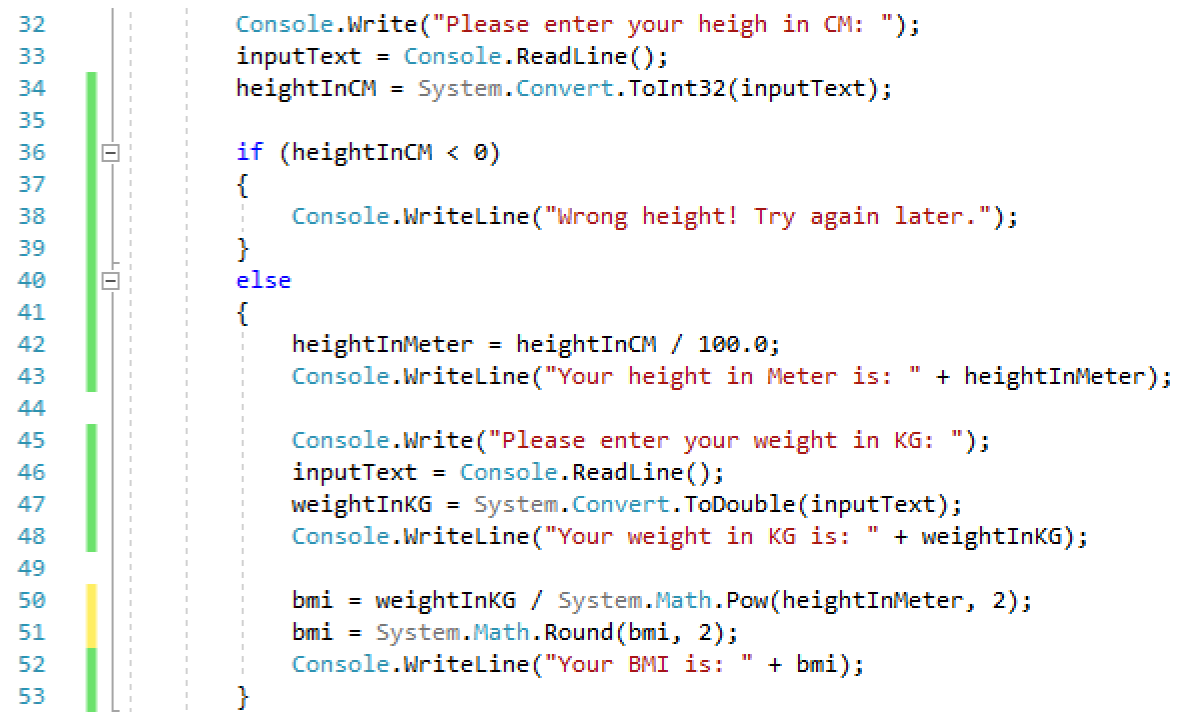
C:\Program Files\dotnet\dotnet.exe

```
Result = 6
Task completed...
```

Conditional statements

If statements

- Are used to check one or more conditions.
- Can change the flow of control to a specific part in code.
- Can include Boolean logic operators...
  - And operator: && [e.g., if (a>10 && b<5)]
  - Or operator: | |
  - Exclusive Or operator: ^
  - Not/negation operator: ! [e.g., if (!x), if (y != 5)]
  - More at: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/boolean-logical-operators

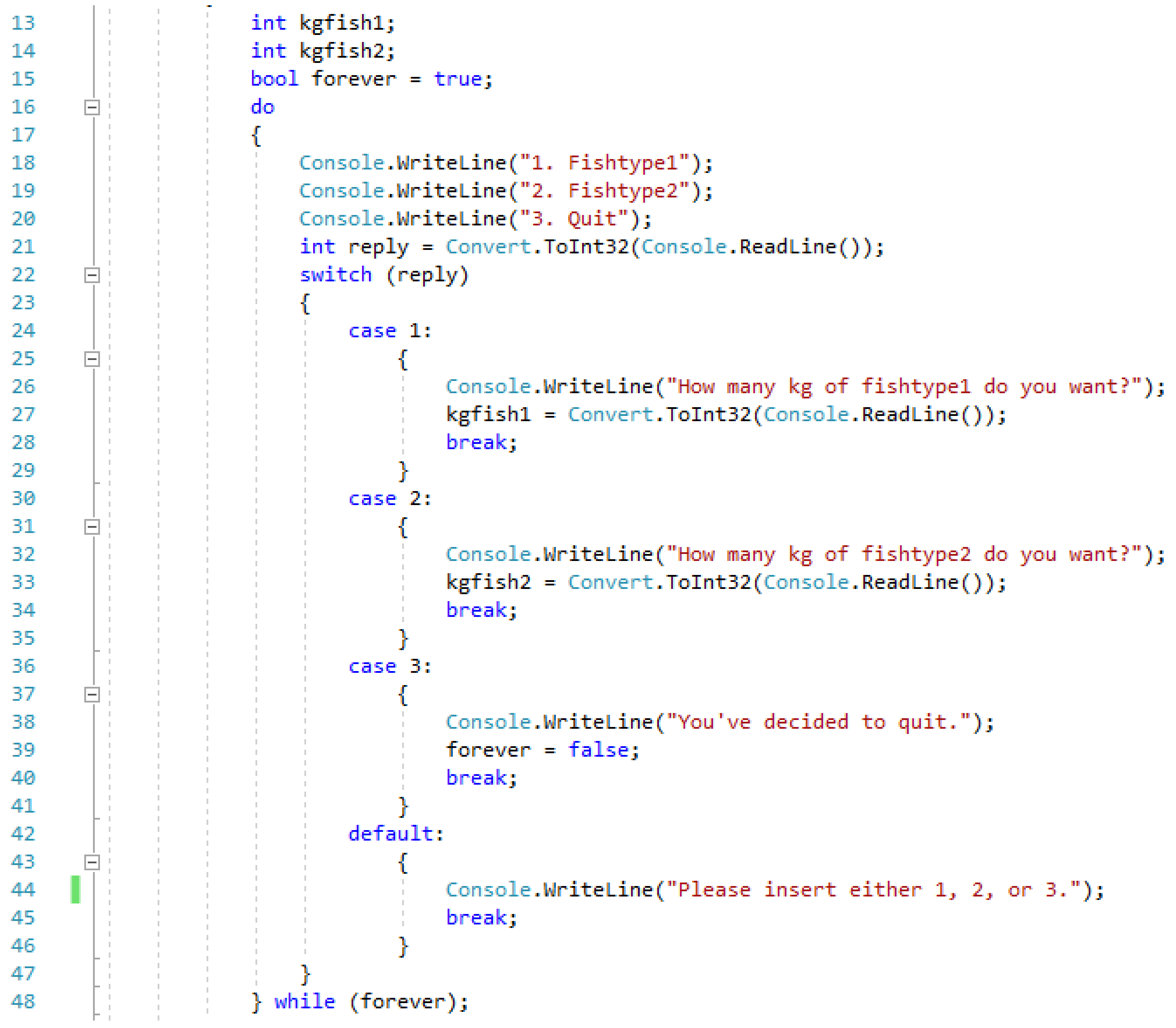


```
Console.Write("Please enter your heigh in CM: ");
inputText = Console.ReadLine();
heightInCM = System.Convert.ToInt32(inputText);

if (heightInCM < 0)
{
    Console.WriteLine("Wrong height! Try again later.");
}
else
{
    heightInMeter = heightInCM / 100.0;
    Console.WriteLine("Your height in Meter is: " + heightInMeter);

    Console.Write("Please enter your weight in KG: ");
    inputText = Console.ReadLine();
    weightInKG = System.Convert.ToDouble(inputText);
    Console.WriteLine("Your weight in KG is: " + weightInKG);

    bmi = weightInKG / System.Math.Pow(heightInMeter, 2);
    bmi = System.Math.Round(bmi, 2);
    Console.WriteLine("Your BMI is: " + bmi);
}
```

Conditional statements

Switch statements

- To be used when there are **several valid choices** to select from.
  - The **break** command makes it possible to only run one block; otherwise, more tests are done for the next cases.
  - The **default** block is run if none of the other cases is selected, this is optional.

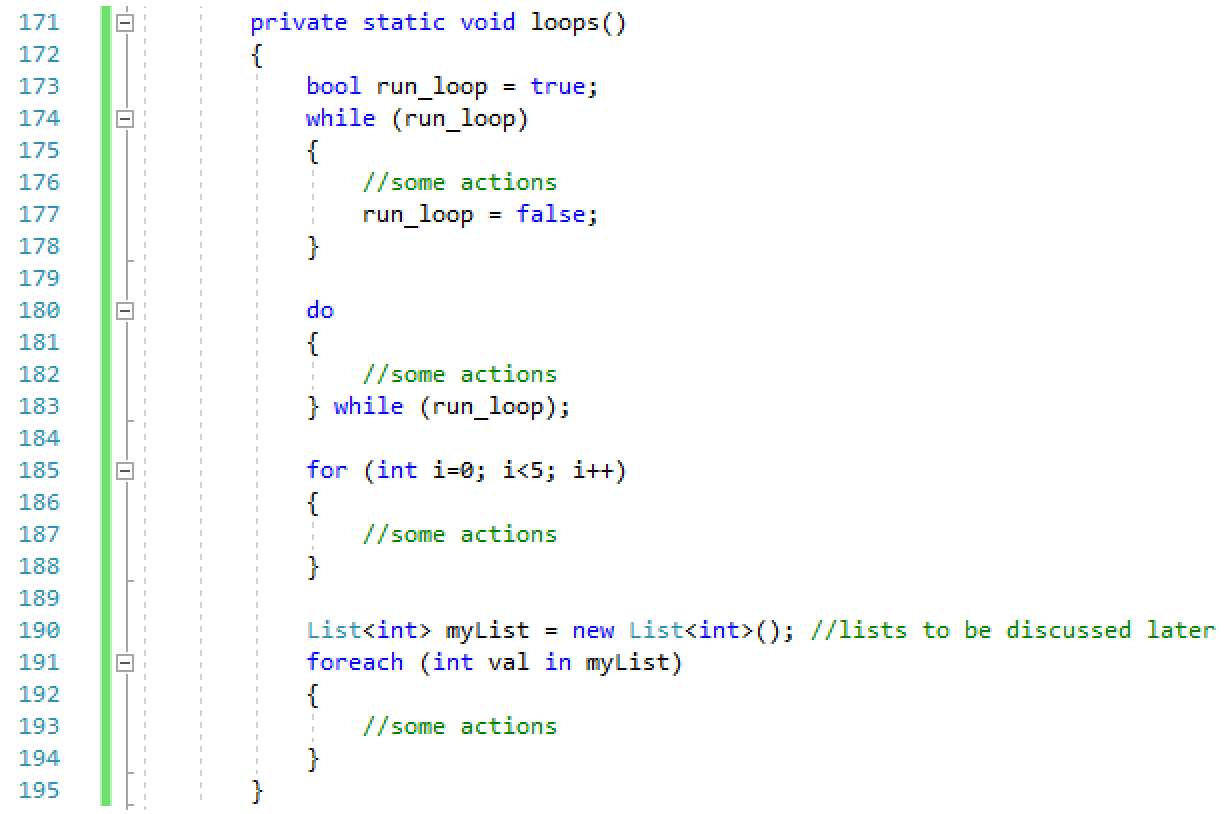


```
int kgfish1;
int kgfish2;
bool forever = true;
do
{
    Console.WriteLine("1. Fishtype1");
    Console.WriteLine("2. Fishtype2");
    Console.WriteLine("3. Quit");
    int reply = Convert.ToInt32(Console.ReadLine());
    switch (reply)
    {
        case 1:
            {
                Console.WriteLine("How many kg of fishtype1 do you want?");
                kgfish1 = Convert.ToInt32(Console.ReadLine());
                break;
            }
        case 2:
            {
                Console.WriteLine("How many kg of fishtype2 do you want?");
                kgfish2 = Convert.ToInt32(Console.ReadLine());
                break;
            }
        case 3:
            {
                Console.WriteLine("You've decided to quit.");
                forever = false;
                break;
            }
        default:
            {
                Console.WriteLine("Please insert either 1, 2, or 3.");
                break;
            }
    }
} while (forever);
```

Loops

4 types in C#.Net

- To be used when there is a need for some actions to be repeated.
  - The **while** loop – runs until the condition holds
  - The **do-while** loop – runs until the condition holds (at least once as it post-checks the condition)
  - The **for** loop – runs for a specific number of times specified using a condition
  - The **foreach** loop – runs for every item within a list or an array



```
private static void loops()
{
    bool run_loop = true;
    while (run_loop)
    {
        //some actions
        run_loop = false;
    }

    do
    {
        //some actions
    } while (run_loop);

    for (int i=0; i<5; i++)
    {
        //some actions
    }

    List<int> myList = new List<int>(); //lists to be discussed later
    foreach (int val in myList)
    {
        //some actions
    }
}
```

WHAT ELSE CAN CHANGE THE FLOW OF CONTROL...?

ERRORS AND ERROR DETECTION

The effect of errors

Errors in code...

- Can be of two types:
    - Compile-time (syntax errors)
    - Run-time (exceptions)

- Can change the flow of control.
- Can reduce program robustness and data integrity and accuracy.
- Need to be caught and handled...
    - **Error detection**, to identify when and what error has occurred.
    - **Error handling**, to implement the right procedure to handle the unexpected situation and correct for the error.

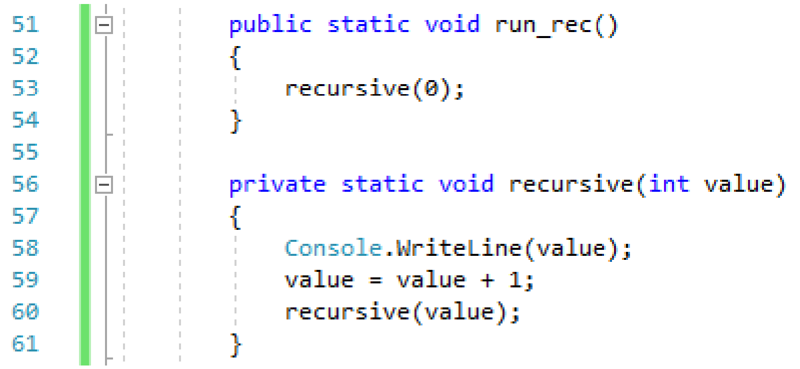
I THOUGHT YOU SAID THIS APPLICATION WAS BUG-FREE!
OH YES, THEY'RE FREE! WE DON'T CHARGE EXTRA FOR THEM!
LANG ©2014
SQLSERVERCENTRAL.COM

Image source: https://www.qamadness.com/how-much-do-software-bugs-really-cost/

Error detection scenarios

Potential errors: Example 1

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in OutOfMemoryException.
  - It will result in OverflowException.
  - It will result in IndexOutOfRangeException.
  - It will result in StackOverflowException.



```
public static void run_rec()
{
    recursive(0);
}

private static void recursive(int value)
{
    Console.WriteLine(value);
    value = value + 1;
    recursive(value);
}
```

Error detection scenarios

Potential errors: Example 1 (cont.)

- The code will result in **StackOverflowException**
  - For each call of the function to itself (recursion), a value is added to **stack**
  - The recursive function calls itself infinitely
  - Too many (unbounded) recursions will make the stack full and thus, overflow
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.stackoverflowexception?view=net-5.0

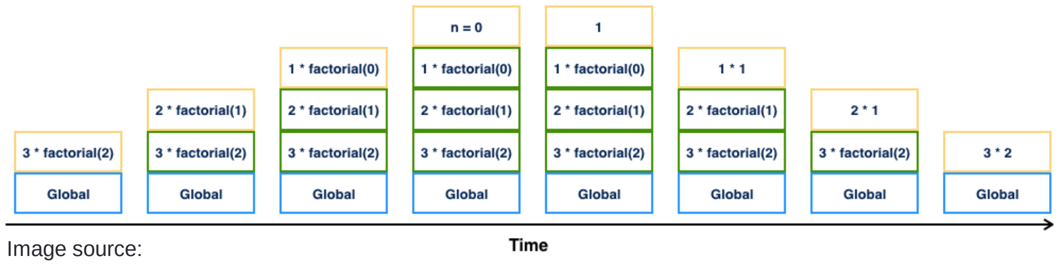
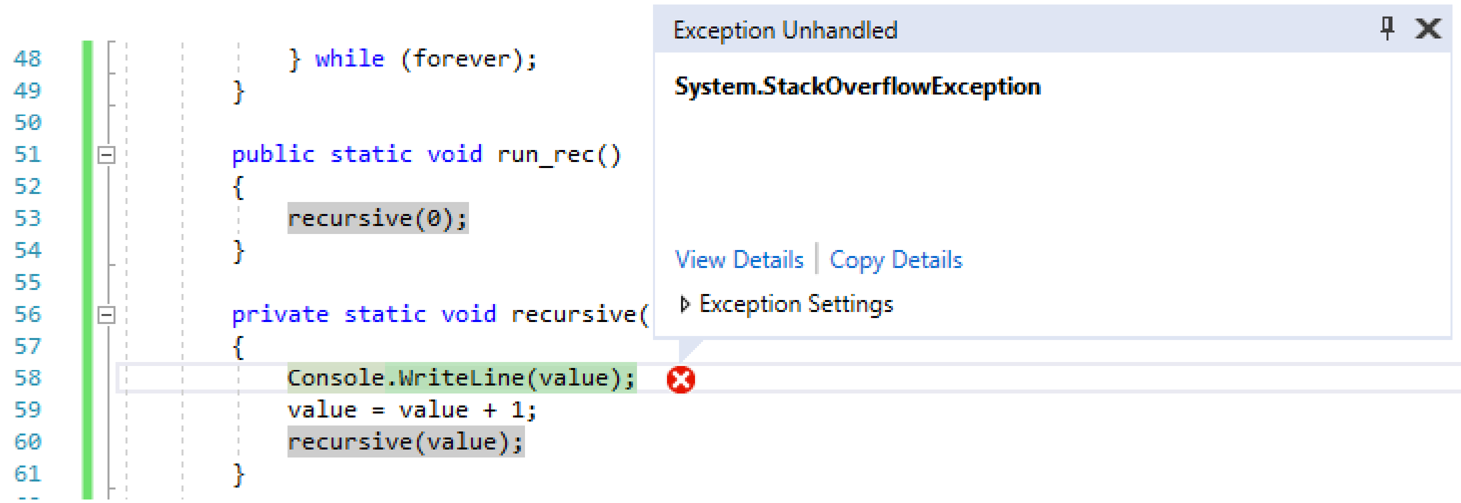


Image source: http://www.thinkingincrowd.me/2016/06/06/How-to-avoid-Stack-overflow-error-on-recursive/



```
        } while (forever);
    }

    public static void run_rec()
    {
        recursive(0);
    }

    private static void recursive(
    {
        Console.WriteLine(value);
        value = value + 1;
        recursive(value);
    }
```

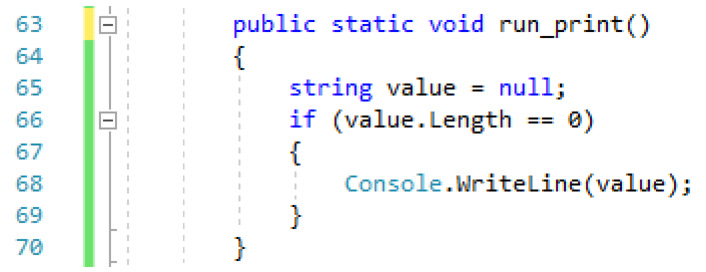
Exception Unhandled

System.StackOverflowException

Error detection scenarios

Potential errors: Example 2

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in OutOfMemoryException.
  - It will result in InvalidCastException.
  - It will result in DivideByZeroException.
  - It will result in StackOverflowException.

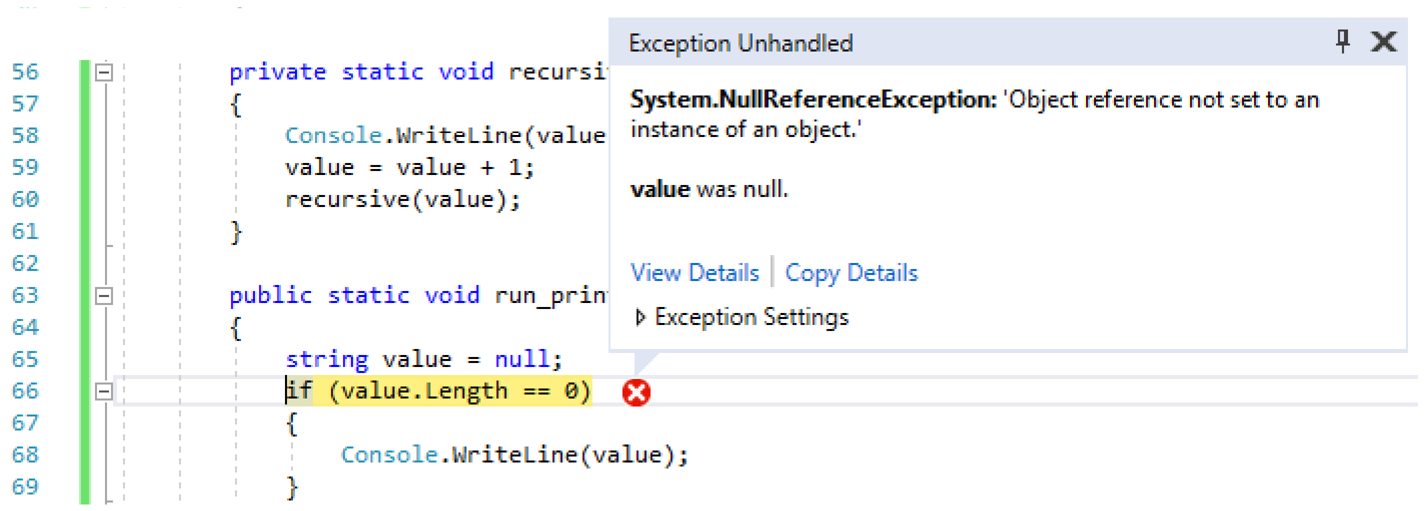


```
public static void run_print()
{
    string value = null;
    if (value.Length == 0)
    {
        Console.WriteLine(value);
    }
}
```

Error detection scenarios

Potential errors: Example 2 (cont.)

- The code will result in **NullReferenceException**
  - It indicates that you are trying to access a thing that does not exist (i.e., null object reference).
  - The string variable does not point to any location in memory (this sort of memory is called **heap**).
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.nullreferenceexception?view=net-5.0



```
        private static void recursi
        {
            Console.WriteLine(value
            value = value + 1;
            recursive(value);
        }

        public static void run_prin
        {
            string value = null;
            if (value.Length == 0)
            {
                Console.WriteLine(value);
            }
```

Exception Unhandled

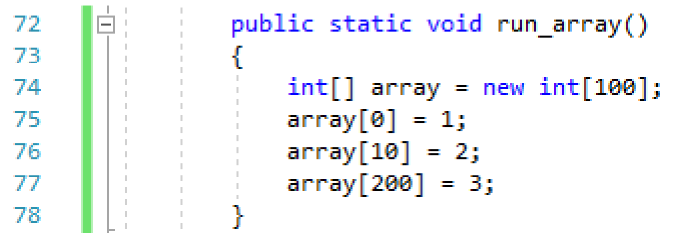
System.NullReferenceException: 'Object reference not set to an instance of an object.'

value was null.

Error detection scenarios

Potential errors: Example 3

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in OutOfMemoryException.
  - It will result in OverflowException.
  - It will result in IndexOutOfRangeException.
  - It will result in FileNotFoundException.

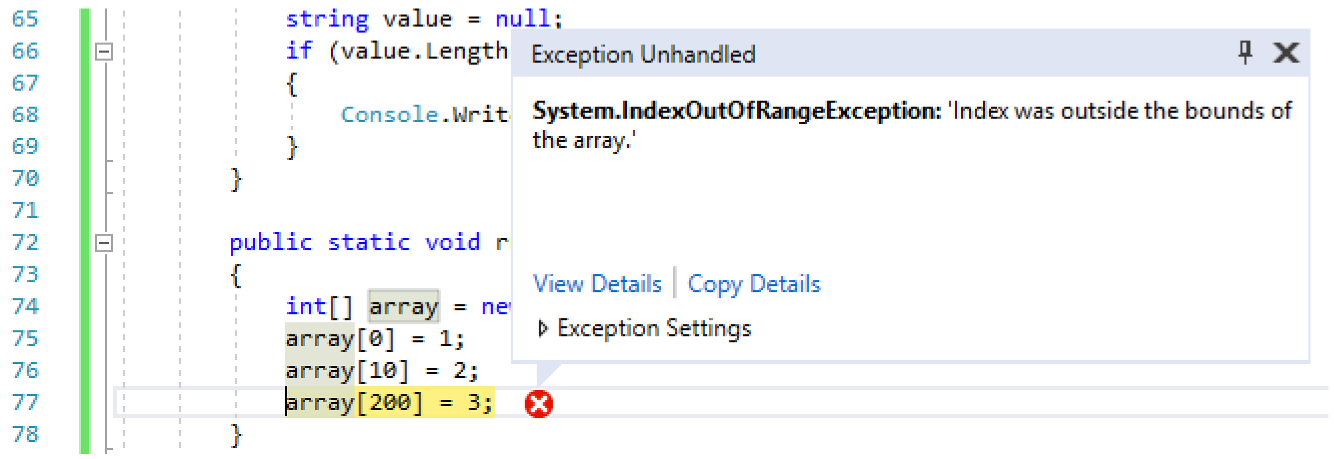


```
public static void run_array()
{
    int[] array = new int[100];
    array[0] = 1;
    array[10] = 2;
    array[200] = 3;
}
```

Error detection scenarios

Potential errors: Example 3 (cont.)

- The code will result in **IndexOutOfRangeException**
  - This error happens in C# programs that use arrays when a statement tries to access an element at an index greater than the maximum allowable index.
  - Here, for an array of 100 elements, you can access array[0] through array[99] only.
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.indexoutofrangeexception?view=net-5.0



```
        string value = null;
        if (value.Length
        {
            Console.Writ
        }
    }

    public static void r
    {
        int[] array = ne
        array[0] = 1;
        array[10] = 2;
        array[200] = 3;
    }
```

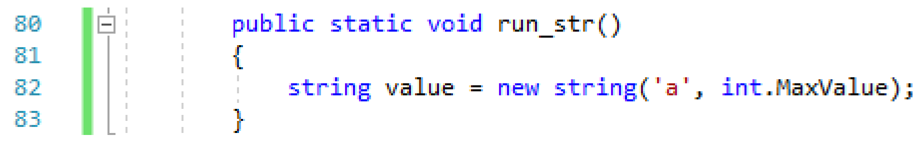
Exception Unhandled

System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'

Error detection scenarios

Potential errors: Example 4

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in OutOfMemoryException.
  - It will result in OverflowException.
  - It will result in IndexOutOfRangeException.
  - It will result in FileNotFoundException.

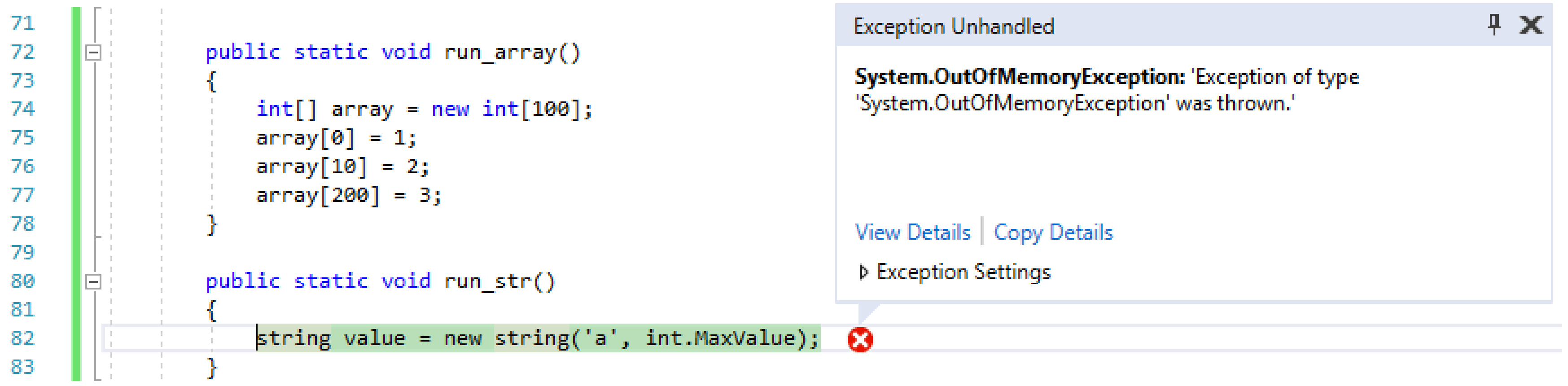


```
public static void run_str()
{
    string value = new string('a', int.MaxValue);
}
```

Error detection scenarios

Potential errors: Example 4 (cont.)

- The code will result in **OutOfMemoryException**
  - Note: Memory is limited!
  - This error can occur during any allocation call at runtime, when program requests for RAM, but free memory is not available.
  - This program attempts to allocate a string that is extremely large and would occupy four gigabytes of memory, that is not possible in this case.
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.outofmemoryexception?view=net-5.0



```
public static void run_array()
{
    int[] array = new int[100];
    array[0] = 1;
    array[10] = 2;
    array[200] = 3;
}

public static void run_str()
{
    string value = new string('a', int.MaxValue);
}
```

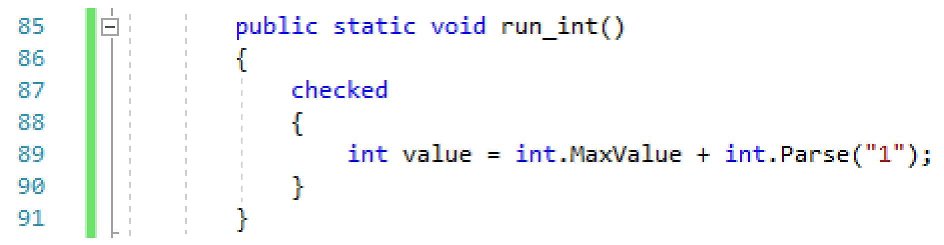
Exception Unhandled

System.OutOfMemoryException: 'Exception of type 'System.OutOfMemoryException' was thrown.'

Error detection scenarios

Potential errors: Example 5

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in OutOfMemoryException.
  - It will result in OverflowException.
  - It will result in IndexOutOfRangeException.
  - It will result in FileNotFoundException.



```
public static void run_int()
{
    checked
    {
        int value = int.MaxValue + int.Parse("1");
    }
}
```

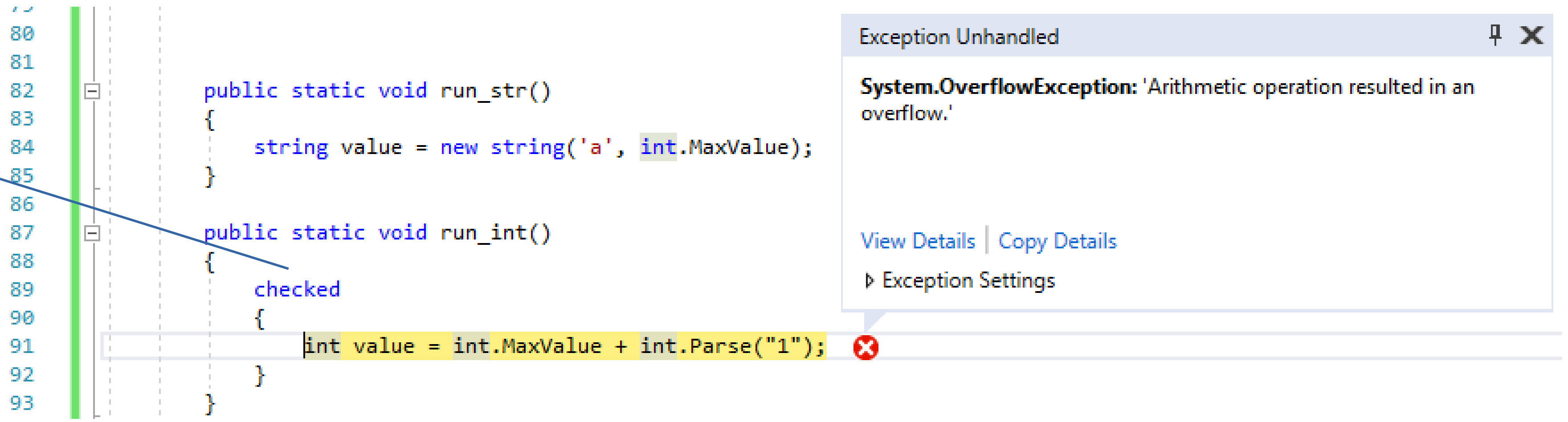
Error detection scenarios

Potential errors: Example 5 (cont.)

- The code will result in **OverflowException**
  - An OverflowException is only thrown in a **checked** context at runtime.
  - It alerts you to an integer overflow; a situation where the number becomes too large to be represented in bytes.
  - An integer takes four bytes. More bytes would be needed to represent the desired number in this case (the number is greater than the maximum value property of **int**).
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.overflowexception?view=net-5.0

In a **checked** C# block, arithmetic overflows raise exceptions. If you use **unchecked**, overflow will result in truncated values (high-order bits).

For more on **checked**, see: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/checked



```
        public static void run_str()
        {
            string value = new string('a', int.MaxValue);
        }

        public static void run_int()
        {
            checked
            {
                int value = int.MaxValue + int.Parse("1");
            }
        }
```

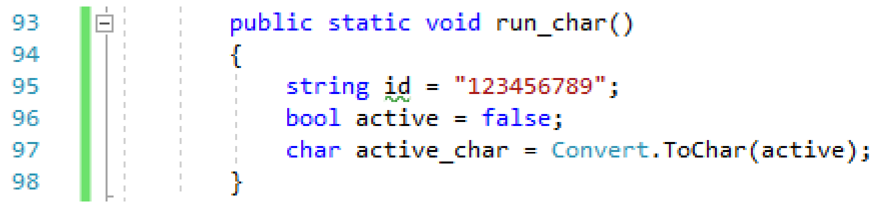
Exception Unhandled

System.OverflowException: 'Arithmetic operation resulted in an overflow.'

Error detection scenarios

Potential errors: Example 6

- Does the code below result in any error? If so, identify a scenario and the nature of the problem in the code.
  - The code is fine, no errors.
  - It will result in NullReferenceException.
  - It will result in ArgumentException.
  - It will result in InvalidCastException.
  - It will result in IndexOutOfRangeException.
  - It will result in StackOverflowException.

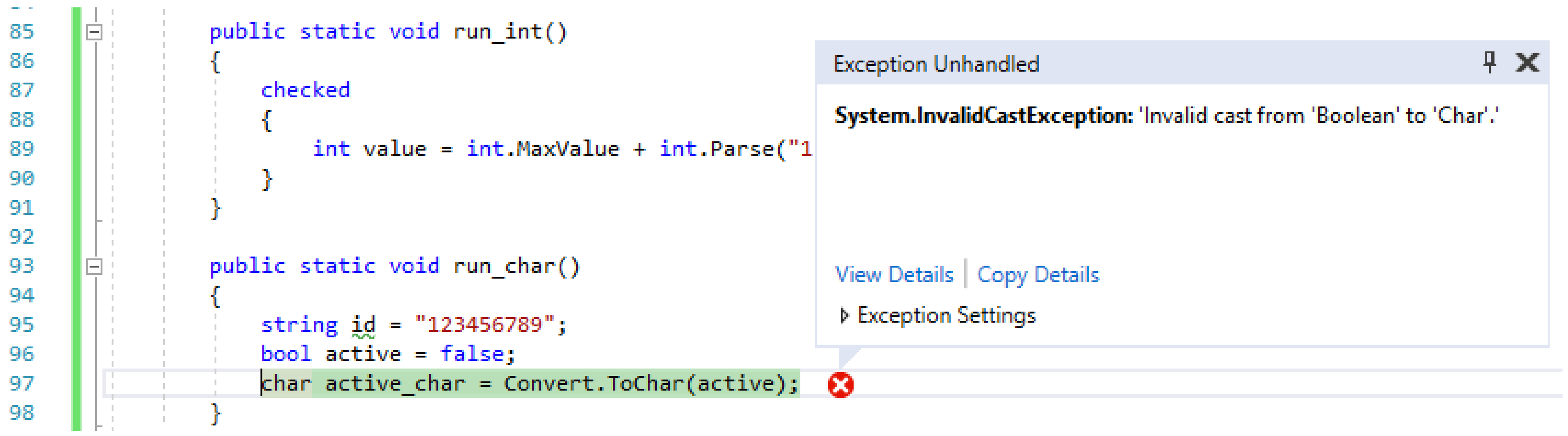


```
public static void run_char()
{
    string id = "123456789";
    bool active = false;
    char active_char = Convert.ToChar(active);
}
```

Error detection scenarios

Potential errors: Example 6 (cont.)

- The code will result in **InvalidCastException**
  - This error occurs when an explicit cast is applied, but the type is not in the same path of the type hierarchy.
  - It is generated by the runtime when a statement tries to cast one reference type to a reference type that is not compatible.
  - More at: https://docs.microsoft.com/en-us/dotnet/api/system.invalidcastexception?view=net-5.0



```
public static void run_int()
{
    checked
    {
        int value = int.MaxValue + int.Parse("1
    }
}

public static void run_char()
{
    string id = "123456789";
    bool active = false;
    char active_char = Convert.ToChar(active);
}
```

Exception Unhandled

System.InvalidCastException: 'Invalid cast from 'Boolean' to 'Char'.'

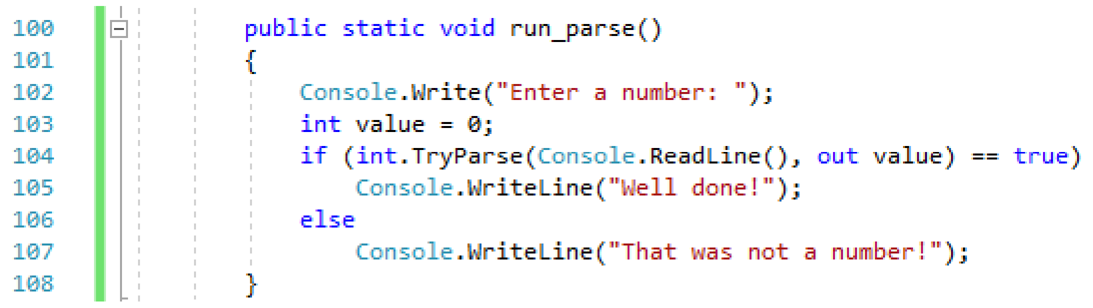
ERRORS IDENTIFIED, WHAT NEXT...?

ERROR HANDLING

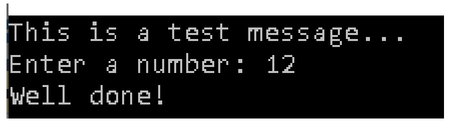
Using specific methods

With some data types...

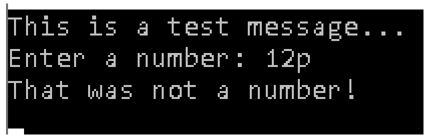
- It is possible to use methods such as:
  - **Parse()**: Tries to parse and will raise an exception if unsuccessful.
  - **TryParse()**: Tries to parse and will only return a Boolean, true for successful and false for unsuccessful. There will be no exception if parsing is unsuccessful.



```
public static void run_parse()
{
    Console.Write("Enter a number: ");
    int value = 0;
    if (int.TryParse(Console.ReadLine(), out value) == true)
        Console.WriteLine("Well done!");
    else
        Console.WriteLine("That was not a number!");
}
```



```
This is a test message...
Enter a number: 12
Well done!
```

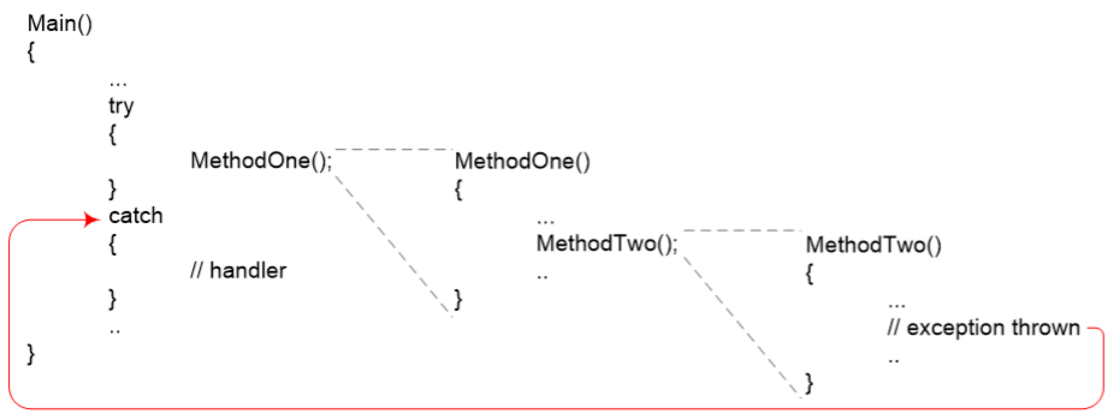


```
This is a test message...
Enter a number: 12p
That was not a number!
```

Try-catch exception handling

C# try-catch blocks

- When an **error** is detected…
    - An **error object** is created with some information about the specific error. The error object is **thrown** directly to the error handling routine in .Net. The error will change the flow of control in the middle of code where the exception or the error has taken place. The code lines after this position in the code will not execute.
    - The error handling routine will try to find a corresponding resolution for the exception within a relevant catch block. If unsuccessful, the program will **crash**!
    - More at: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/clr-inside-out-handling-corrupted-state-exceptions



```
Main()
{
        ...
        try
        {
                MethodOne();
        }
        catch
        {
                // handler
        }
        ..
}
```

```
MethodOne()
{
        ...
        MethodTwo();
        ..
}
```

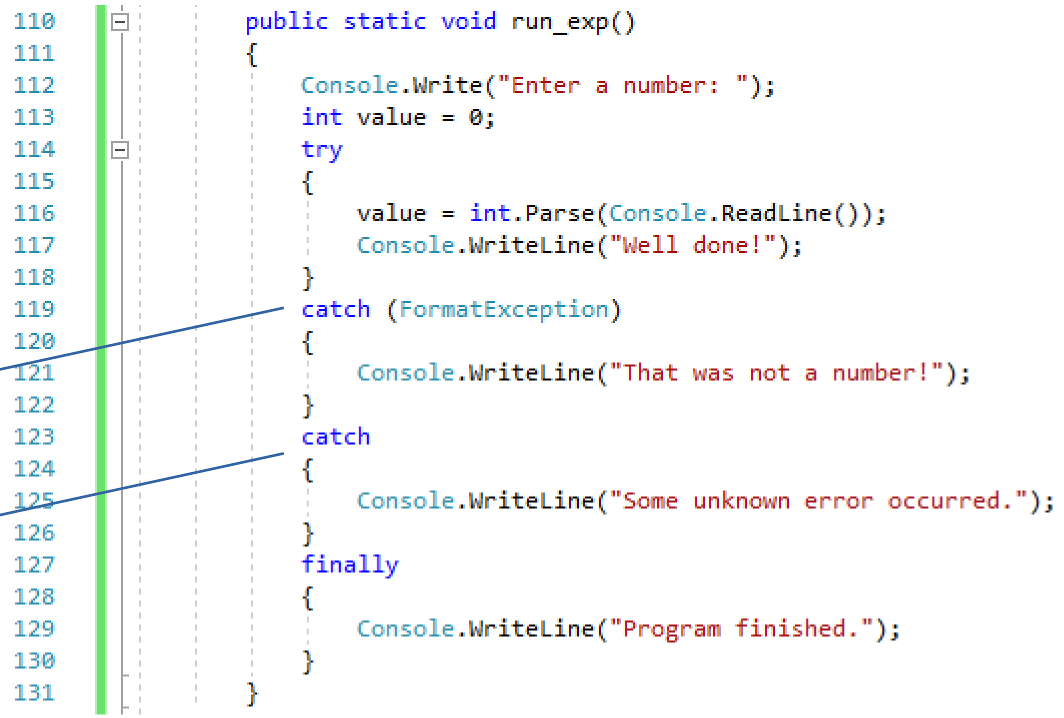
```
MethodTwo()
{
        ...
        // exception thrown
        ..
}
```

Try-catch exception handling

C# try-catch blocks (cont.)

- Exceptions can be caught using try-catch-finally blocks…
    - The **try block** contains code that may throw an exception.
    - The **catch blocks** handle different exceptions that are thrown (zero or more).
    - The **finally block** executes regardless of exception or not.
    - The **finally block** is usually used to free up resources.

- **Note:** The order of catch blocks is important. A general catch with no specific exception type will prevent execution of more specific exception handler catch blocks.



```
public static void run_exp()
{
    Console.Write("Enter a number: ");
    int value = 0;
    try
    {
        value = int.Parse(Console.ReadLine());
        Console.WriteLine("Well done!");
    }
    catch (FormatException)
    {
        Console.WriteLine("That was not a number!");
    }
    catch
    {
        Console.WriteLine("Some unknown error occurred.");
    }
    finally
    {
        Console.WriteLine("Program finished.");
    }
}
```

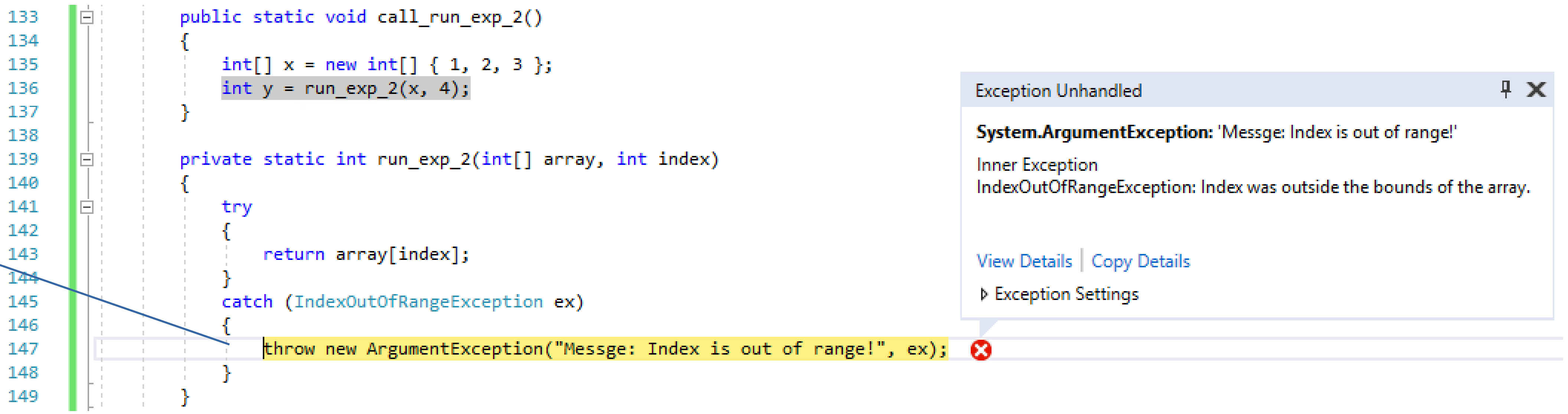
The specific exception of **FormatException** is caught and handled.

When no type given, any exception can be caught and handled.

Try-catch exception handling

Throw exceptions

- Exceptions can be thrown actively using the **throw** command…
  - throw new exceptiontype();
  - throw new exceptiontype(message);
  - throw new exceptiontype(message, inner_exception);



```
public static void call_run_exp_2()
{
    int[] x = new int[] { 1, 2, 3 };
    int y = run_exp_2(x, 4);
}

private static int run_exp_2(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (IndexOutOfRangeException ex)
    {
        throw new ArgumentException("Messge: Index is out of range!", ex);
    }
}
```

Exception Unhandled

System.ArgumentException: 'Messge: Index is out of range!'

Inner Exception
IndexOutOfRangeException: Index was outside the bounds of the array.

An **ArgumentException** object is created and thrown with a custom message.

Try-catch exception handling

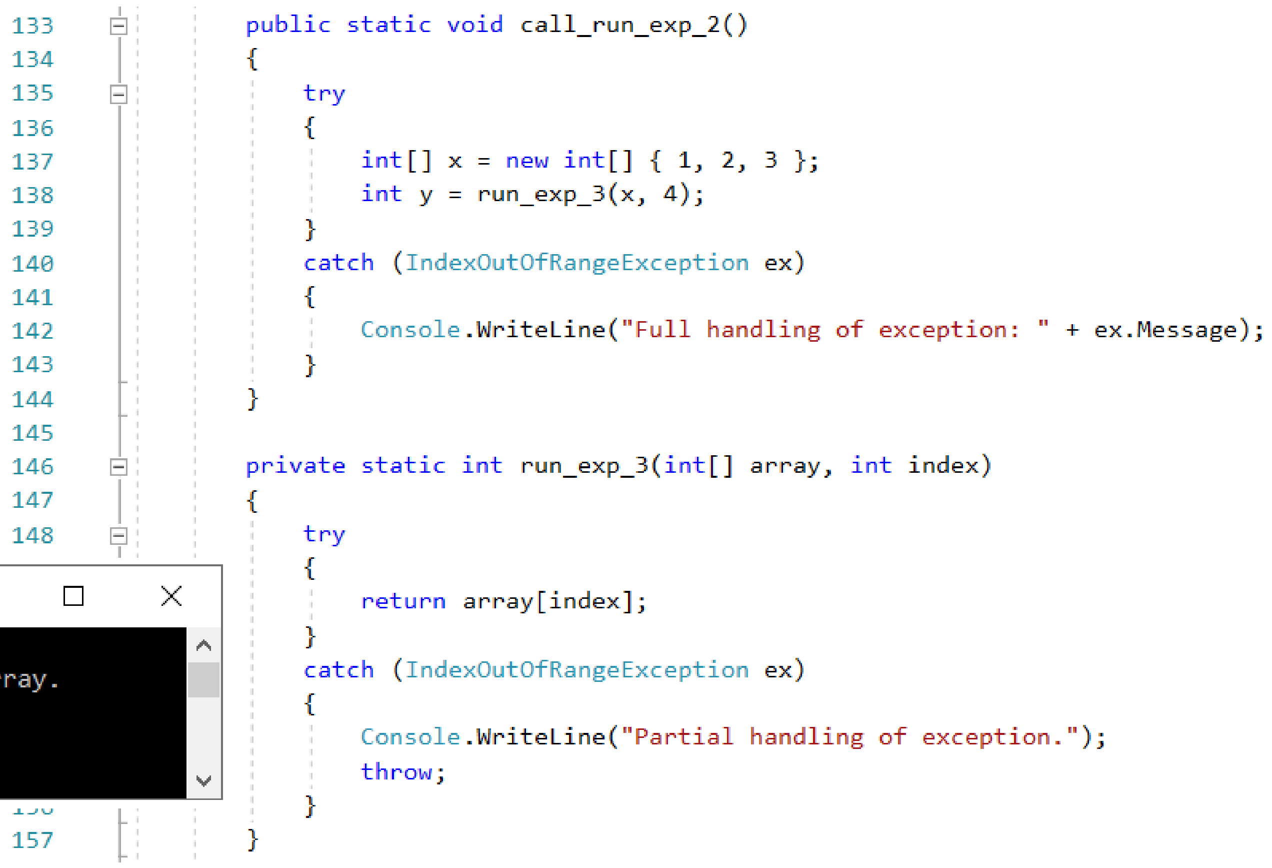
Throw exceptions

- Guidelines
  - Use exceptions to *notify other parts* of the program about errors that should not be ignored.
  - Throw an exception only for conditions that are *truly exceptional*.
  - Do not use an exception to *shift the responsibility* for the error to someone else.
  - Avoid throwing exceptions in *constructors/destructors* or catch them in the same place.
  - Include in the exception message *all the information* that led to the exception.
  - Do not use exceptions *as return types* instead of throwing them.
  - Avoid *empty catch* blocks. Why?!

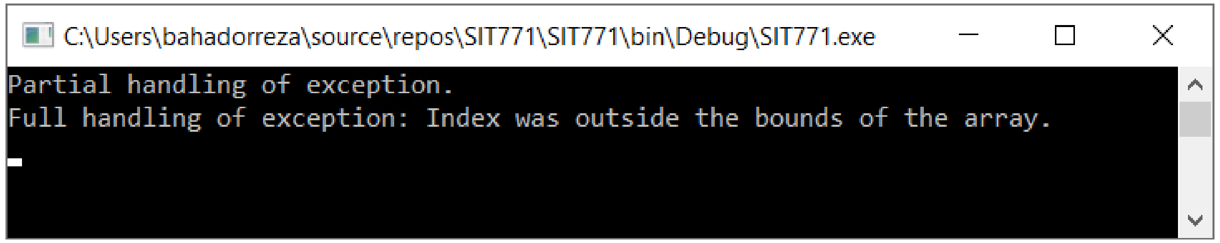
Try-catch exception handling

Re-throw exceptions

- Proper way: Use **throw** without any arguments, to re-throw the exact same exception
- This is used to…
  - handle exceptions by the caller stack.
  - more effectively and completely handle an exception (in addition to partial handling in a local function).
  - free up resources that may still be in use.



```
public static void call_run_exp_2()
{
    try
    {
        int[] x = new int[] { 1, 2, 3 };
        int y = run_exp_3(x, 4);
    }
    catch (IndexOutOfRangeException ex)
    {
        Console.WriteLine("Full handling of exception: " + ex.Message);
    }
}

private static int run_exp_3(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (IndexOutOfRangeException ex)
    {
        Console.WriteLine("Partial handling of exception.");
        throw;
    }
}
```



C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe

```
Partial handling of exception.
Full handling of exception: Index was outside the bounds of the array.
```

.Net exception classes

Standard exceptions

- The following is the list of exception classes in .Net
  - **StackOverflowException** – the call stack cannot grow any larger;
  - **OutOfMemoryException** – the system has run out of memory;
  - **NullReferenceException** – an attempt is made to access an attribute/operation of an object when the reference is set to null;
  - **ArgumentException** – one or more arguments were invalid;
  - **FileNotFoundException** – specified file does not exist;
  - **InvalidCastException** – a data type casting is not valid (usually because the types are unrelated);
  - **DivideByZeroException** – attempt made to divide by zero;
  - **IndexOutOfRangeException** – array index is out of range;
  - **OverflowException** – converting a value, such as with the Convert object, results in the loss of data, e.g., attempting to convert the value 123456 to a byte (which has the range of 0-255).

.Net exception classes

Purpose-built exception classes

- The exception classes provided by .Net will often not be adequate – this is not unusual. Instead, it is possible to create your own exception classes by deriving the class representing the exception from the **Exception** (base) class via the **inheritance** mechanism.
- More on this later...

Epilogue

EXPERIENCE IS THE NAME EVERYONE GIVES TO THEIR MISTAKES…

OSCAR WILDE