Scopes (not the sniper version)

Scopes

Scopes is a very important concept in C#. Imagine an application like World of Warcraft. The code must be huge and contain millions of variables!

You might imagine that after a short amount of time, all variable names might be in use already. To avoid conflicts here, variables only exist in a limited scope in the application, where ever it is needed.

Like street names

It is similar to street names.. apparently, Ringvägen is the most common street name in sweden. Does this cause packages to Ringvägen 1 to arrive at the wrong address constantly?

Thankfully not, since each Ringvägen is uniquely defined within its zip code, or “Scope” if you think in terms of programming!

Scope delimiter

Scopes in C# are started using { and ended using }.

```
{          // Scope Start
    int age = 2;
}          // Scope End
```

Nested Scopes

Scopes can be nested, every { must have a matching }

Each } closes a the last opened, unclosed {.

```
{       // Scope A Start

        {       // Scope B Start

        }       // Scope B End


        {       // Scope C Start

        }       // Scope C End

}       // Scope A End
```

Variable Scope

A variable is valid within its scope only, that is, between the previous { and the matching }.

```
{ // Variable `a` Scope Start
  int a = 5; // `a` is declared here, its scope is the closest, previous `{`
  {
    Console.WriteLine(a); // `a` is still valid within the nested scope
  }
} // Variable `a` Scope End
```

Not accessible

Outside of the scope, you can NOT access the variable!

```
{
  { // Variable `a` Scope Start
    int a = 5;
  } // Variable `a` Scope End
  Console.WriteLine(a); // Error, a is not defined in this scope.
}
```

Can’t declare with same name

You can NOT declare two variables with the same name within the same scope:

```
{
  int z = 2;
  int z = 3; // ERROR: Variable named `z` is defined already
}
```

Not even if they are nested

Also not, if they are nested:

```
{
  int z = 2;
  {
    int z = 3; // ERROR: Variable named `z` is defined already
  }
}
```

But you can still use existing

But of course, you can still assign new values to existing variables:

```
{
  int z = 2;
  z = 3; // This is fine, we assign a new value to the existing variable.
}
```

Goal - Input in different types

- Ask the user for a number and assign the result to a variable of type string named input.
- Increment input
- Print input to the Console.
- Ask the user for another number and assign the result to a variable of type int named input.
- Increment input
- Print input to the Console.
- Ask the user for another number and assign the result to a variable of type char named input.
- Increment input
- Print input to the Console.

Remember to use Scopes!

```
Output:Give me a number.
Input:9
Output:91
Output:Give me another number.
Input:9
Output:10
Output:Give me another number.
Input:9
Output::
```