Professional Software Engineering

Patrick Berggold, Andrea Carrara – **Lecturers**

Mohab Hassaan, Hritik Singh – **Tutors**

Chair of Computational Modeling and Simulation

Schedule of the lecture

» Structure of the course

» Programming Paradigms

» **Introduction to C#**

THE C# BASICS

patrick.berggold@tum.de andrea.carrara@tum.de

Prerequisites

» Why is it a good choice?

- **Object-oriented & multi purpose language**

  → Imperative, Functional, Reflective, Generic …
- **Under active development**
- **Plattform independent** (*.Net Core works on Windows, Apple and Linux*)
- **Most professional programs offer .NET API**
- **Siemens NX, Inventor, Revit, Word, Excel, …**
- **Provides user-friendly libraries (.NET framework)**

  → ASP .NET, LINQ, Entity-Framework, WFP …

patrick.berggold@tum.de andrea.carrara@tum.de

C# Language

» (Some) features
  – **Unified type system**
  – **all types share a common base type**
  – **e.g. ToString available for any types**
» Extended class members
  – **properties and events are members**
» Strongly typed
  – **you cannot call a function that's designed to accept an integer with a float**

patrick.berggold@tum.de andrea.carrara@tum.de

General Syntax of c#

```
using System;

public class MyFirstClass {
    public static void Main() {
        Console.Write("Hello World!");
        int i = 0;
        string s = "StringInHyphens";
        CalculateTheWorldFormula(i);
    }
}
```

package

class only language

function definition

function call

variable definition

do not forget the ;

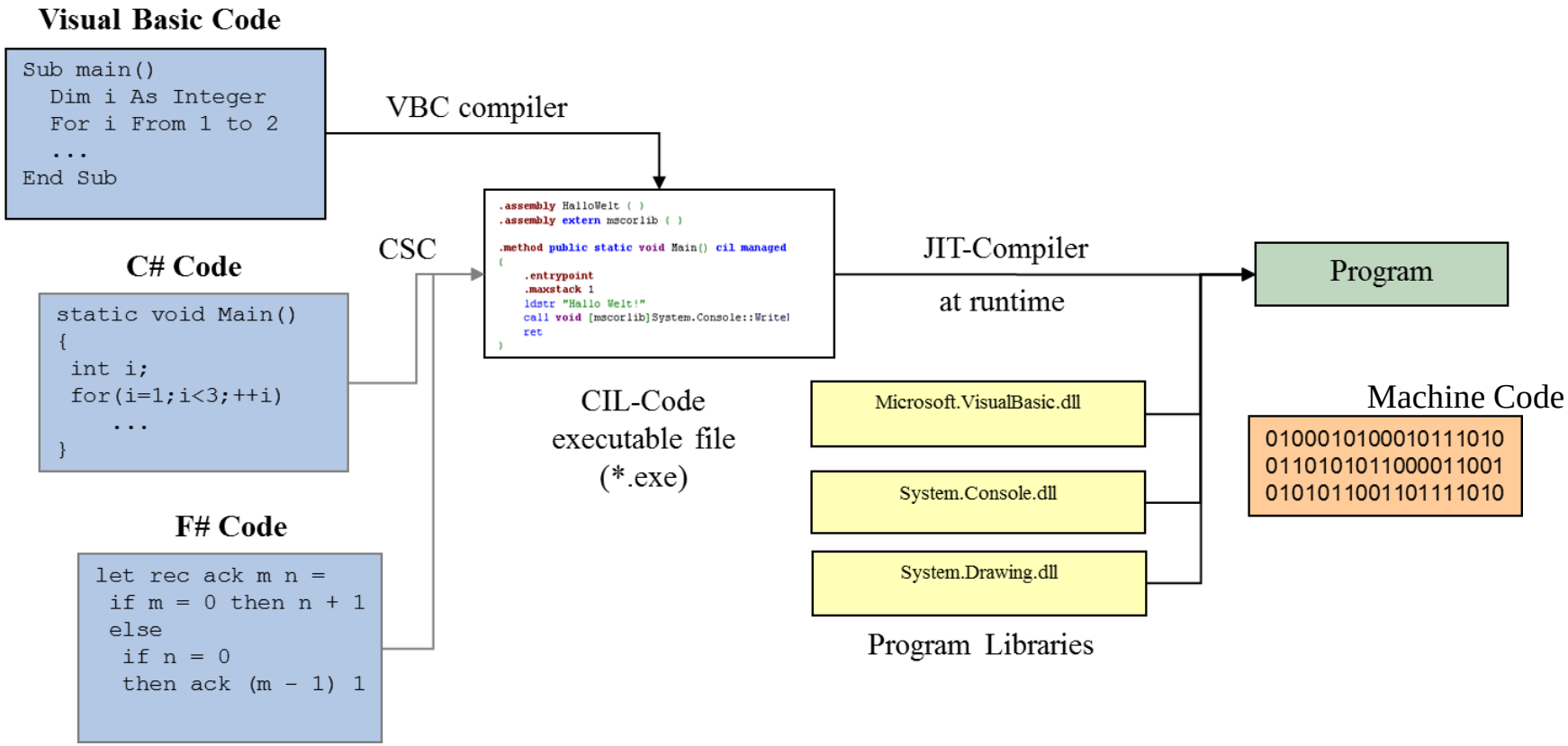
patrick.berggold@tum.de andrea.carrara@tum.de

You Shall Not Use Keywords!

abstract	do	In	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	Out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

patrick.berggold@tum.de andrea.carrara@tum.de

Common Language Infrastructure (CLI)



patrick.berggold@tum.de andrea.carrara@tum.de

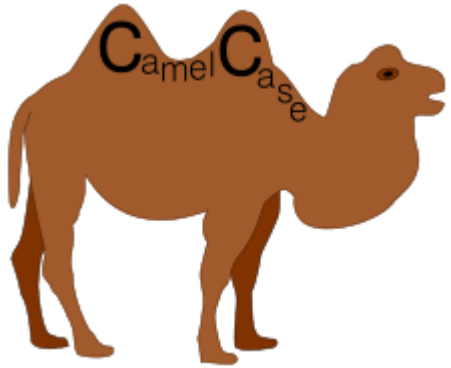
C# Naming Conventions

DO9+

- √ use **PascalCasing** for class names and method names.
- √ use **camelCasing** for method arguments and local variables.
- √ use **noun or noun phrases** to name a class.
- √ **name source files** according to their main classes.

DO NOT

- × use **SCREAMING CAPS** for constants or readonly variables
- × use **Underscores** in identifiers.
- × **suffix** enum names with Enum



patrick.berggold@tum.de andrea.carrara@tum.de

PREREQUISITS

patrick.berggold@tum.de andrea.carrara@tum.de

Check the C#-Tutorials of Microsoft

https://docs.microsoft.com/de-de/dotnet/csharp/tutorials/intro-to-csharp/hello-world

patrick.berggold@tum.de andrea.carrara@tum.de

Things you should know !

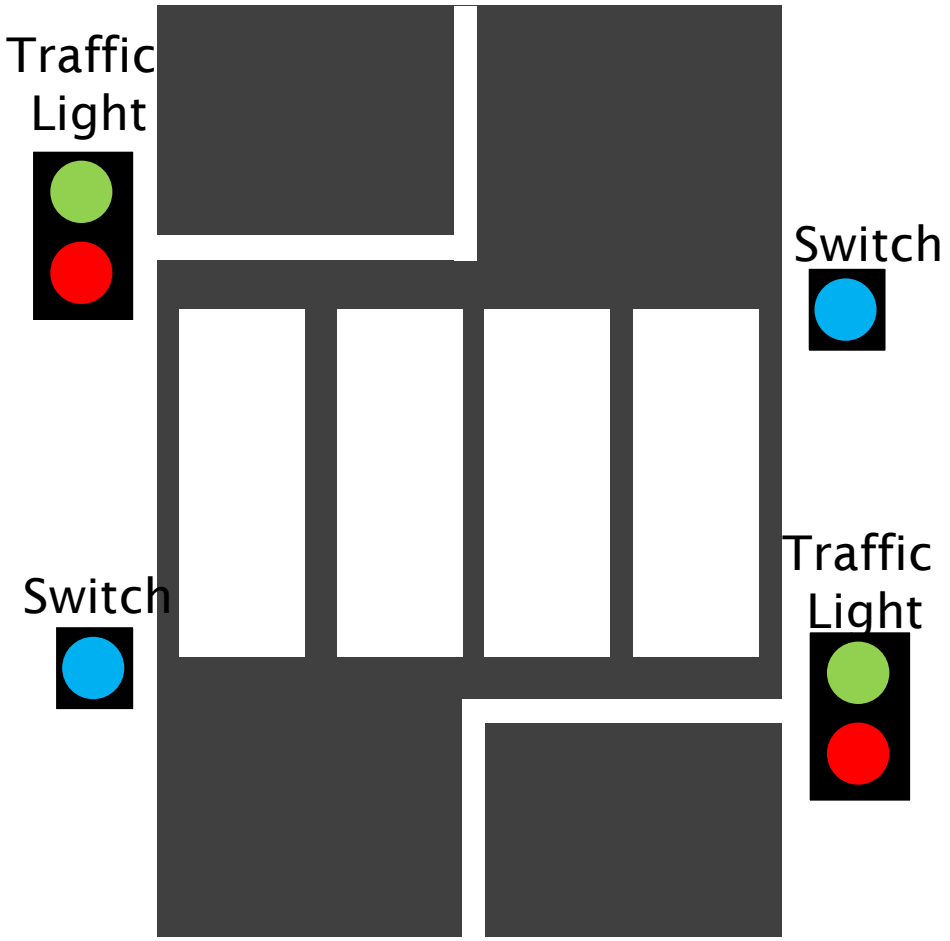
- Control Structures (if, while, for, switch)
- Primitive Datatypes (int ←→ uint, double, string, bool)
- The Usage of functions (passing and returning parameters)
- Frequently used Data Structures (Arrays & Containers)

patrick.berggold@tum.de andrea.carrara@tum.de

EXAMPLE TRAFFIC LIGHT

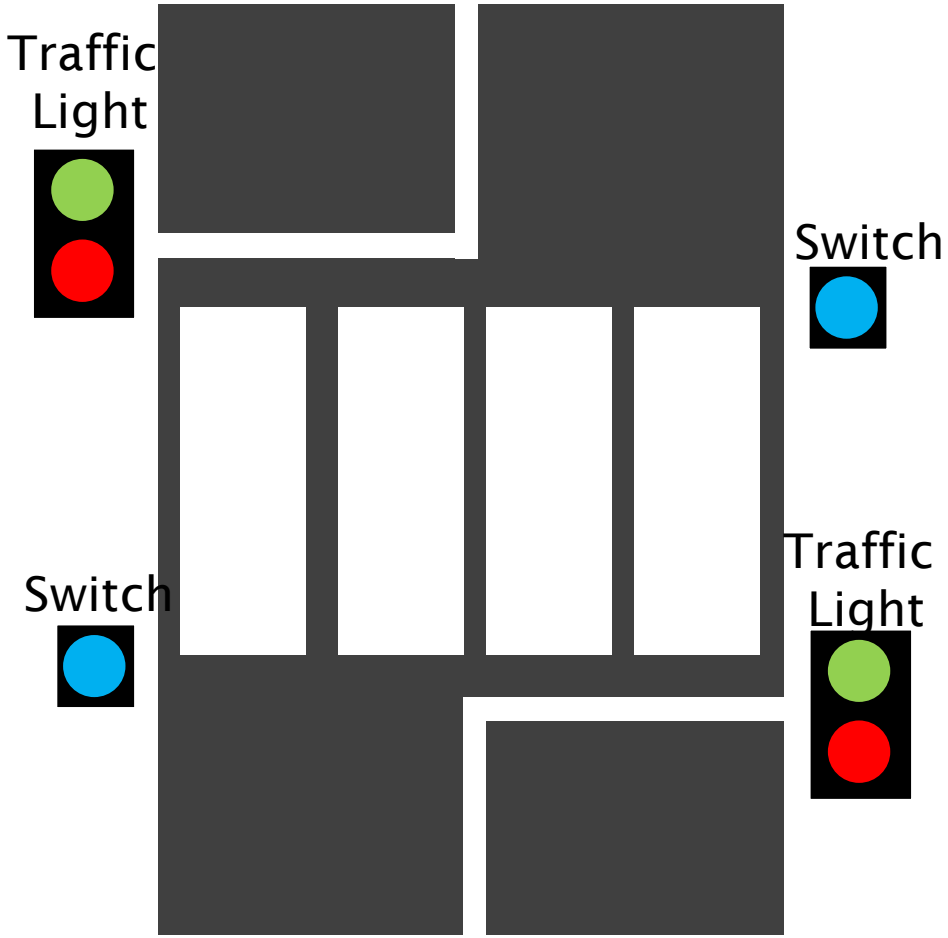
patrick.berggold@tum.de andrea.carrara@tum.de

Traffic light problem definition

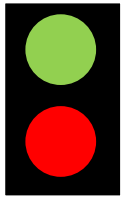


patrick.berggold@tum.de andrea.carrara@tum.de

Traffic light problem definition



Traffic Light

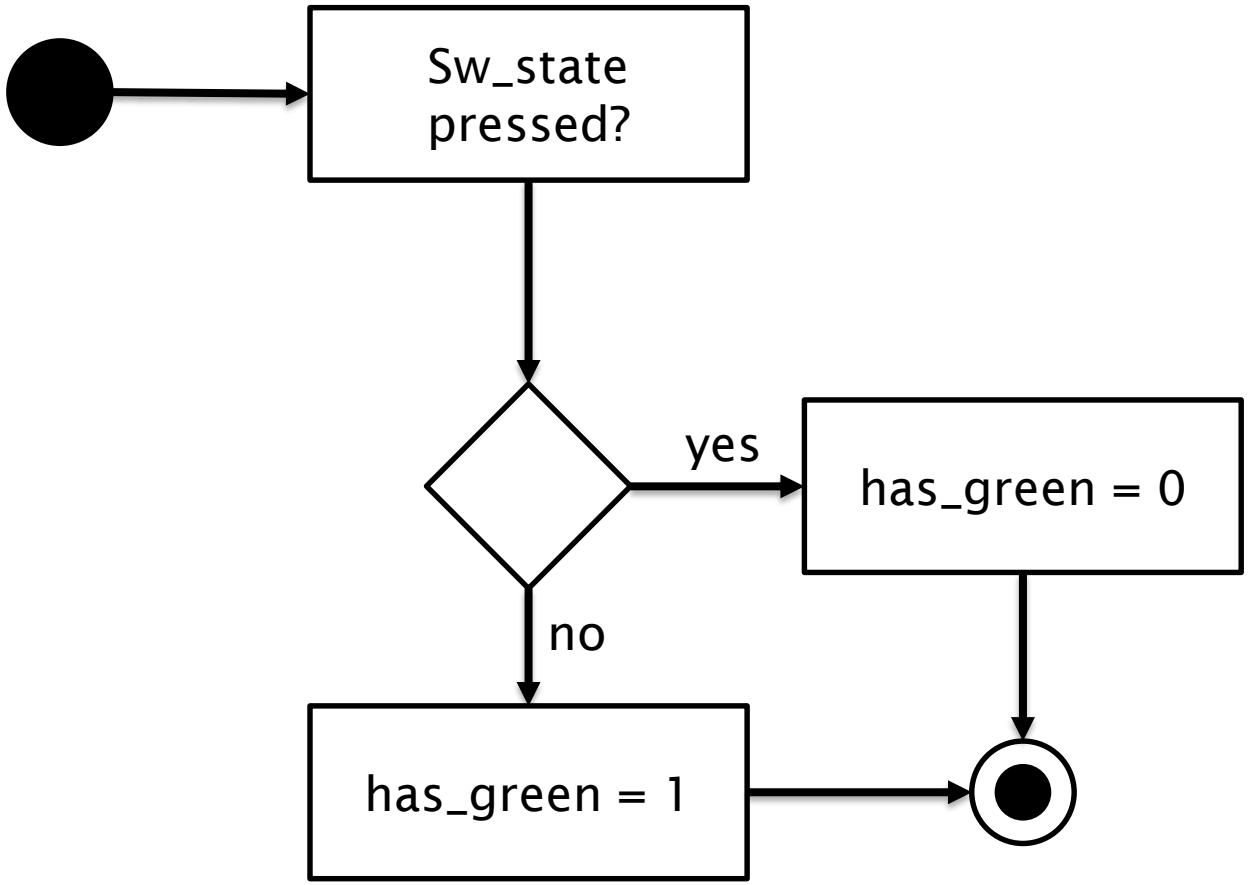


has_green = { 1 / 0

Switch

Switch_state= { true / false

Representation of the problem



patrick.berggold@tum.de andrea.carrara@tum.de

Datatypes

» Value types
  – **bool (true or false)**
  – **char (16-bit Unicode)**
  – **Numeric Types**
  – **Enum**
  – **struct**

» Reference types
  – **object**
  – **DateTime**
  – **and everything derived**

» Strings are immutable !

patrick.berggold@tum.de andrea.carrara@tum.de

Numeric Types

C# type	System type	Suffix	Size	Range
Integral—signed				
sbyte	SByte		8 bits	-2^7 to 2^7-1
short	Int16		16 bits	-2^{15} to $2^{15}-1$
int	Int32		32 bits	-2^{31} to $2^{31}-1$
long	Int64	L	64 bits	-2^{63} to $2^{63}-1$
Integral—unsigned				
byte	Byte		8 bits	0 to 2^8-1
ushort	UInt16		16 bits	0 to $2^{16}-1$
uint	UInt32	U	32 bits	0 to $2^{32}-1$
ulong	UInt64	UL	64 bits	0 to $2^{64}-1$
Real				
float	Single	F	32 bits	$\pm(\sim 10^{-45}$ to $10^{38})$
double	Double	D	64 bits	$\pm(\sim 10^{-324}$ to $10^{308})$
decimal	Decimal	M	128 bits	$\pm(\sim 10^{-28}$ to $10^{28})$

patrick.berggold@tum.de andrea.carrara@tum.de

If - Statement

```
» if( condition )
» {
»     statement block...
» }
» else
» {
»     statement block...
» }
```

```
» if( condition )
»     statement block...;
» else statement block...;
```

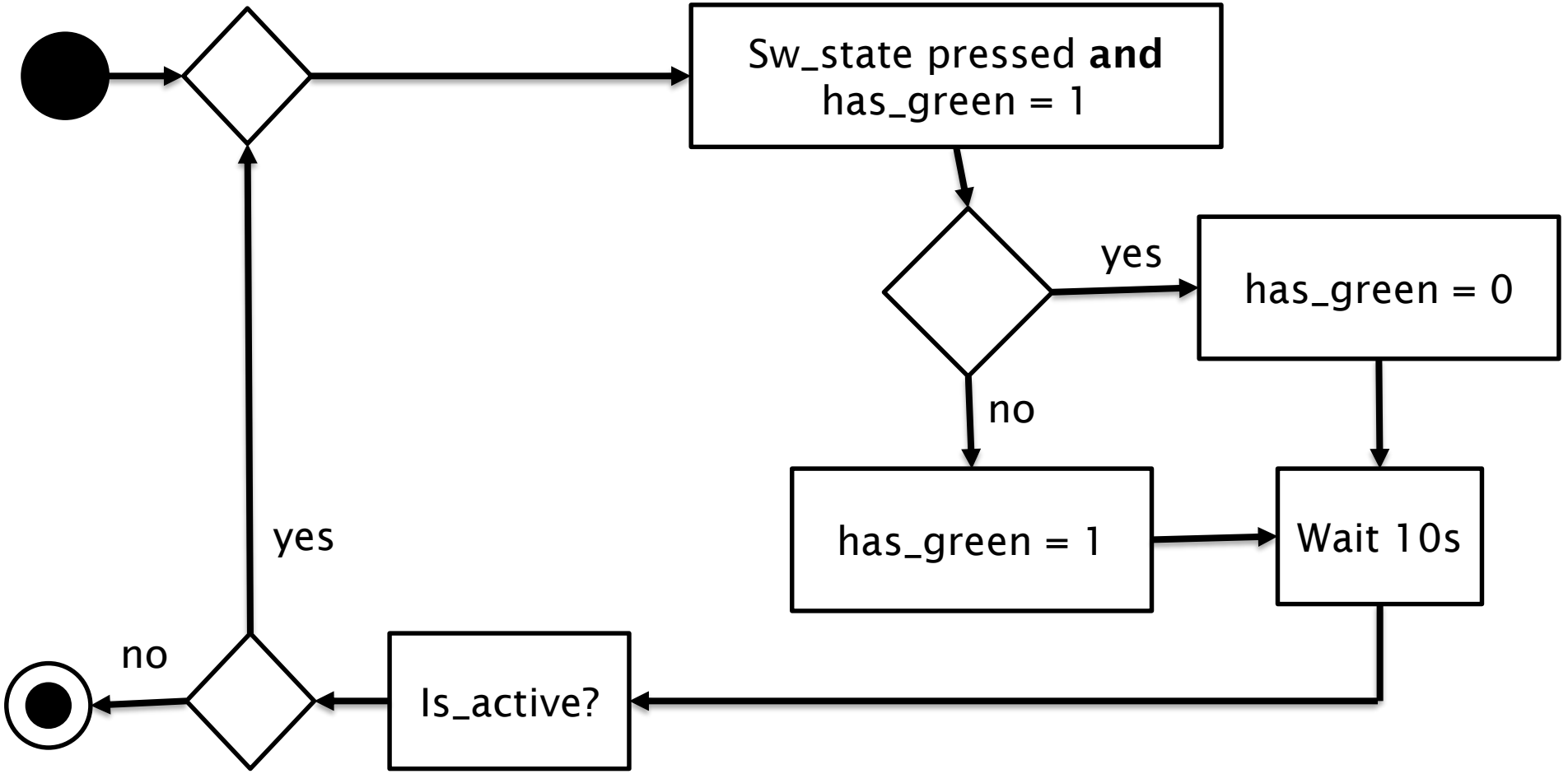
short form (possible if statement block fits in one line)

patrick.berggold@tum.de andrea.carrara@tum.de

If Statements Can Be Cascaded

```
static void TellMeWhatICanDo(int age) {
    if(age >= 40) {
        Console.WriteLine("You can be president!");
    }
    else if(age >= 18) {
        Console.WriteLine("You can Vote!");
    }
    else if(age >= 21) {
        Console.WriteLine("You can drink!");
    }
    else {
        Console.WriteLine("You can wait!");
    }
}
```

Representation of the problem



patrick.berggold@tum.de andrea.carrara@tum.de

While Loop

```
» while (condition)
» {
»     statement block...
» }
```

```
» int i = 0;
» while (i < 3)
» {
»     Console.Write(i++);
» }
```

You can use break, to exit any loop at anytime.

You can use continue, to break one interation (in the loop, if a specified condition occurs, and continues with the next iteration.

patrick.berggold@tum.de andrea.carrara@tum.de

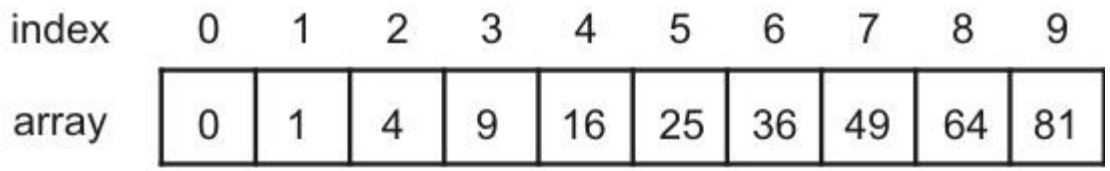
Arrays

- List of values of the same type

  (e.g. list of integer values, list of double values, list of boolean values)
- Access to all values via one variable
- Length of list has to be defined when initializing
- The first element is located at zero !

```
int[] a = new int[10]; // declare int array of length 10
```

```
a[4] = 16;
```



patrick.berggold@tum.de andrea.carrara@tum.de

For Loop

```
for (initializer; condition; iterator )
{
    statement block...
}
```

```
for (int i = 1; i <= 5; i++)
{
   Console.WriteLine(i);
}
```


patrick.berggold@tum.de andrea.carrara@tum.de

Datatypes

» Value types
  - **bool (true or false)**
  - **char (16-bit Unicode)**
  - **Numeric Types**
  - **Enum**
  - **struct**

An `enum` is a special "class" that represents a group of **constants** (unchangeable/read-only variables).

» Reference types
  - **object**
  - **DateTime**
  - **and everything derived**

» Strings are immutable !

patrick.berggold@tum.de andrea.carrara@tum.de

C# OPERATORS

Categories of operators:

1. Arithmetic operators: Perform operations on numbers that are *int*, *double*, *float*, etc.

2. Relational operators: Compare and check the equality of the input objects

3. Logical operators: Compare bits of the given object and always return a *Boolean* result

4. Bitwise operators: Perform operations on individual bits, and the result is also always a bit

5. Assignment operators: allow us to initialize an object with a value or perform specific operations on it

Arithmetic Operators

Operator	Description	Example (A=10, B=20)
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

@https://www.tutorialspoint.com/csharp/csharp_operators.htm

patrick.berggold@tum.de andrea.carrara@tum.de

Relational Operators

Operator	Description	Example (A=10 B=20)
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.

@https://www.tutorialspoint.com/csharp/csharp_operators.htm

Logical & Bitwise Operators

Operator	Description	Example(A=60,B=13) A=0011 1100, B=0000 1101
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = 61, which is 1100 0011

@https://www.tutorialspoint.com/csharp/csharp_operators.htm

patrick.berggold@tum.de andrea.carrara@tum.de

Bitwise Operators - Examples

Bitwise OR (Output = 14 | 11)

14 = 00001110 (In Binary)

11 = 00001011 (In Binary)

15 = 00001111 (In Decimal)

Bitwise AND (Output = 14 | 11)

14 = 00001110 (In Binary)

11 = 00001011 (In Binary)

10 = 00001010 (In Decimal)

Bitwise XOR operator is represented by ^. It performs bitwise XOR operation on the corresponding bits of two operands. If the corresponding bits are same, the result is 0. If the corresponding bits are different, the result is 1.

Bitwise Complement operator is represented by ~. It is a unary operator, i.e. operates on only one operand. The ~ operator inverts each bits i.e. changes 1 to 0 and 0 to 1.

@https://www.tutorialspoint.com/csharp/csharp_operators.htm

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

patrick.berggold@tum.de andrea.carrara@tum.de

Foreach Loop (use if possible)

```
» foreach(var inLoop in iterator )
» {
»     statement block...
» }
```

Switch Statement

```
switch(var) {
        case one:
            // do something
            break;
        case two:
        case three:
            // do something if one is true
            break;
        default:
            // is executed if there is no match
            // no break needed
}
```


patrick.berggold@tum.de andrea.carrara@tum.de

Basic Output

```
Console.WriteLine("Hello World");
Console.Write("Here");
Console.WriteLine("I am.");
```

```
int i = 100;
Console.WriteLine("i = " + i);
```

Casting

» Sometimes you need to convert one type to the other this is referred to as casting

– **There is a difference in converting and casting!**

» To cast a variable in C# you can use:

– **(type) VAR => will throw an error if casting is not possible**

– **VAR as type => returns null if casting is not possible**

→ Only works for reference types !

» Do not get confused casting floating point values!

» Use convert methods when necessary (most of the time!)!

Casting - Example

```
public void CastingExample() {
        double floatingPoint_1 = 1.4;
        double floatingPoint_2 = 1.8;
        int result = 0;

        result = (int) floatingPoint_1;
        //result will now be 1
        result = (int) floatingPoint_2;
        //result will still be 1
        floatingPoint_2 = (double) result;
        // floatingPoint_2 is 1
   }
```


patrick.berggold@tum.de andrea.carrara@tum.de

EXERCISES

patrick.berggold@tum.de andrea.carrara@tum.de