Programming Using C#, Basic Course

Algorithms, Selection and operators

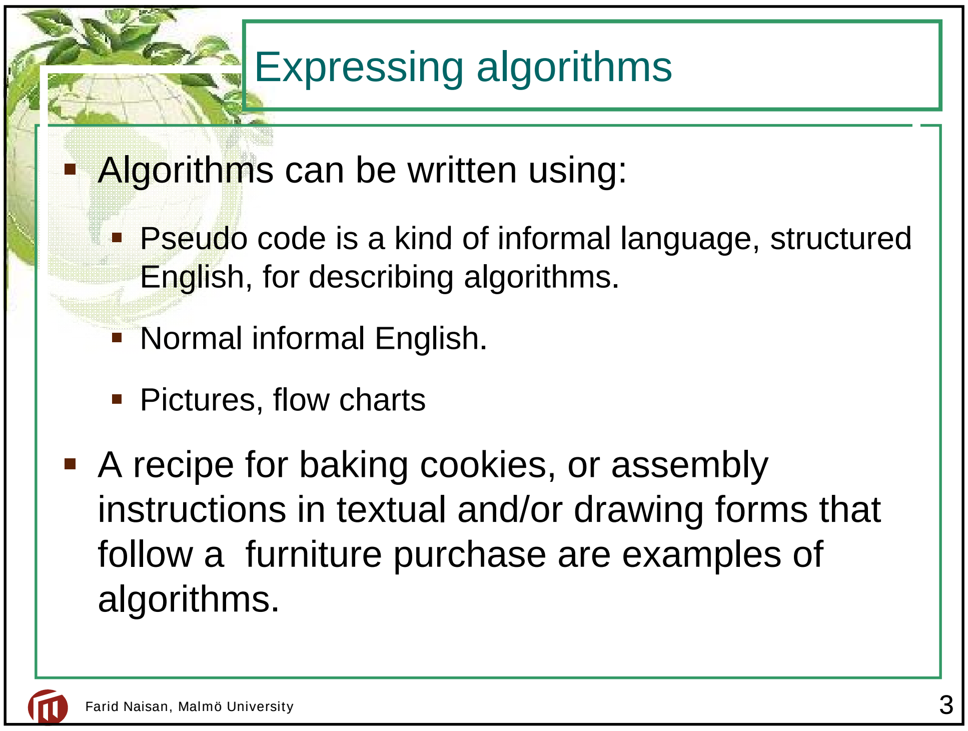
Contents:

Algorithms
Selection algorithm
Comparison, logical and arithmetic operators

Instructor: Farid Naisan, University Lecturer, farid.naisan@mah.se

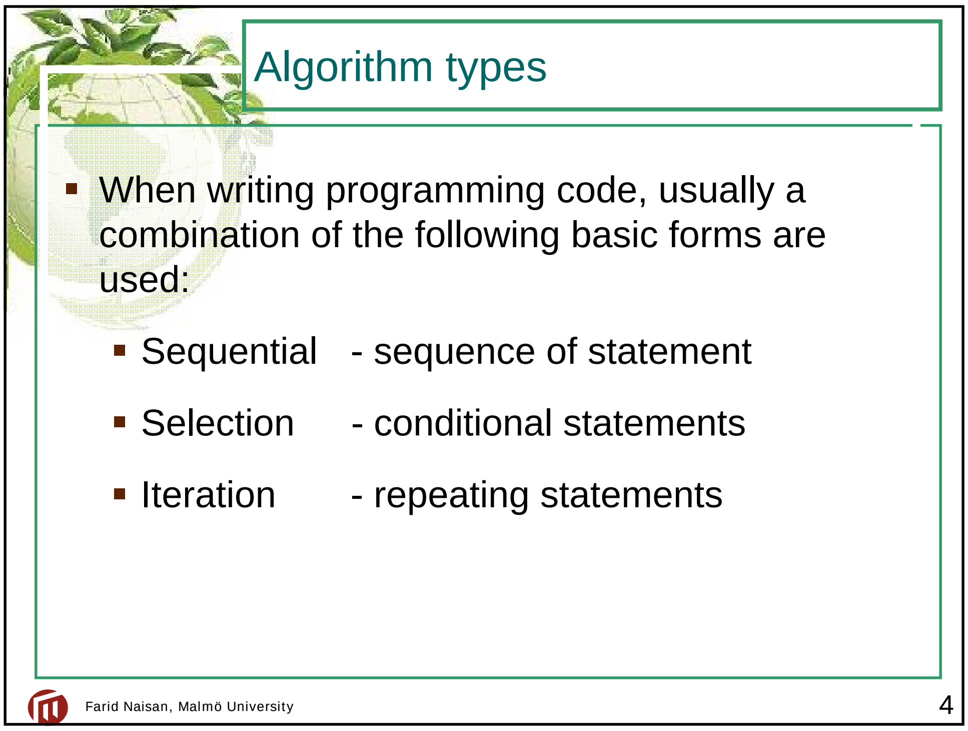
Algorithm

- An algorithm is a well-defined sequence of instructions for completing a task. It has an start and an end state.
- It is simply an approach to solve problems.
- When you program, you always use algorithms. A very simple example:
  - Read input
  - Calculate
  - Present results.



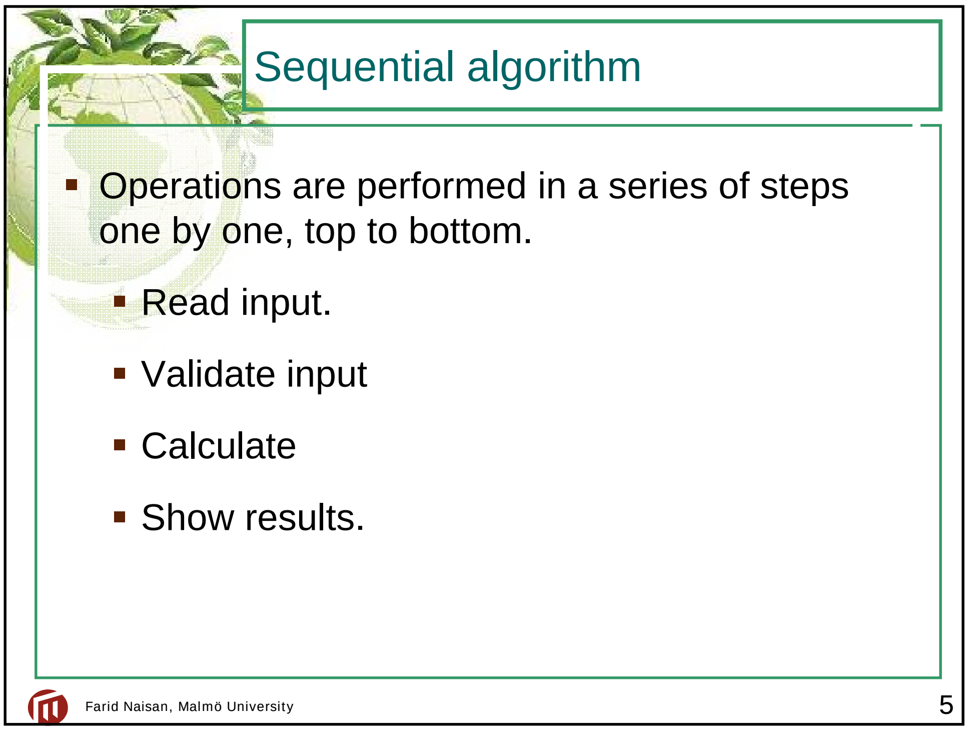
Expressing algorithms

- Algorithms can be written using:
  - Pseudo code is a kind of informal language, structured English, for describing algorithms.
  - Normal informal English.
  - Pictures, flow charts
- A recipe for baking cookies, or assembly instructions in textual and/or drawing forms that follow a furniture purchase are examples of algorithms.



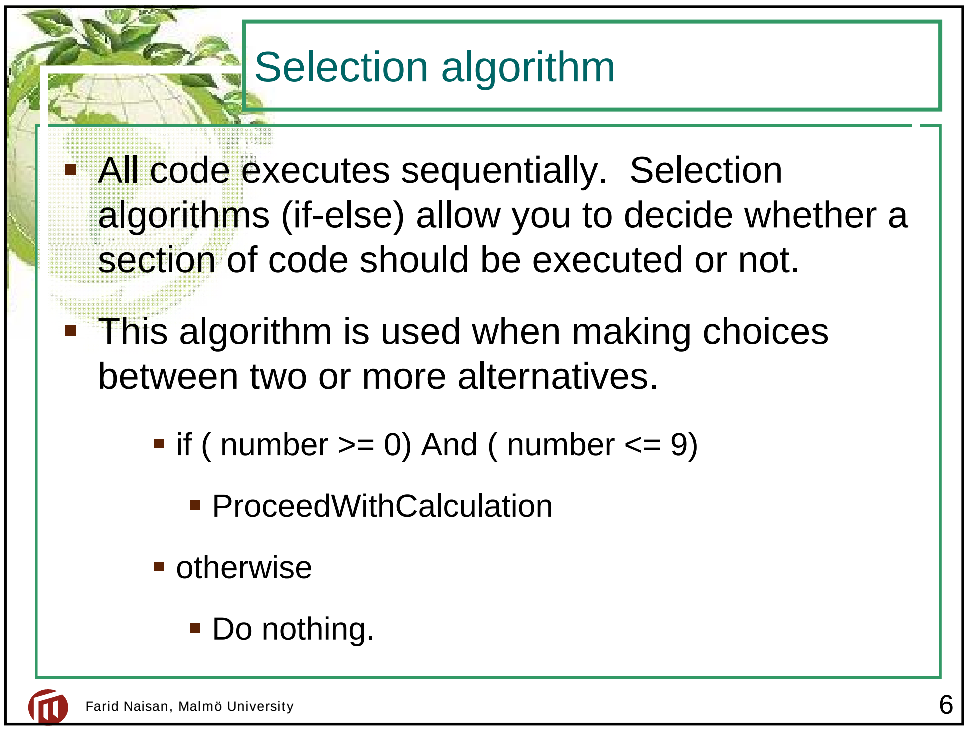
Algorithm types

- When writing programming code, usually a combination of the following basic forms are used:
  - Sequential - sequence of statement
  - Selection - conditional statements
  - Iteration - repeating statements



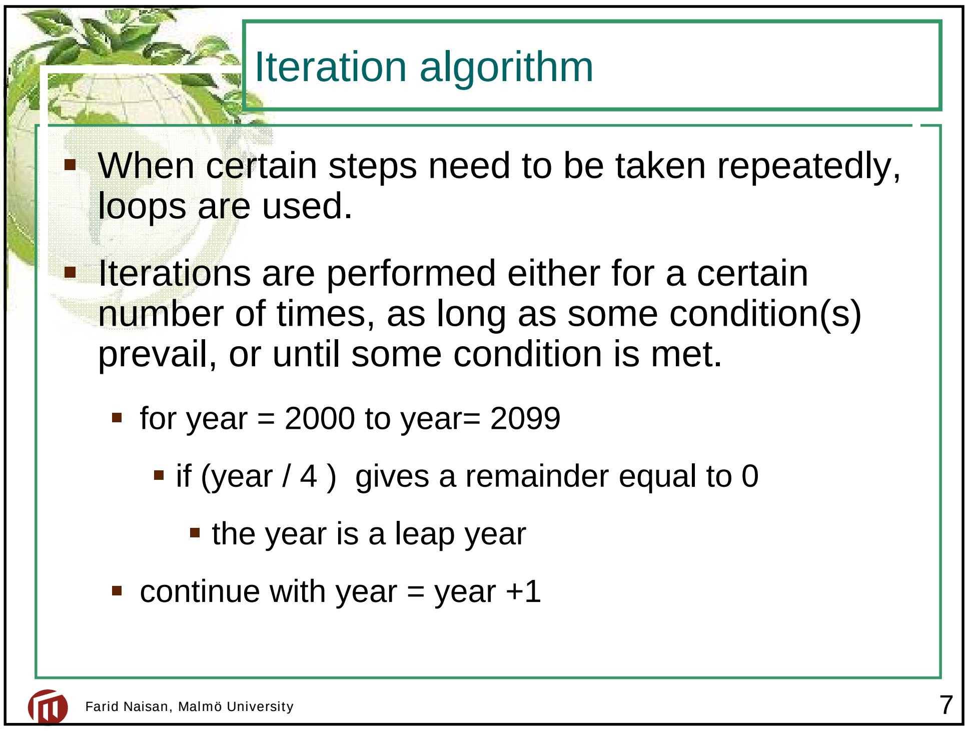
Sequential algorithm

- Operations are performed in a series of steps one by one, top to bottom.
  - Read input.
  - Validate input
  - Calculate
  - Show results.



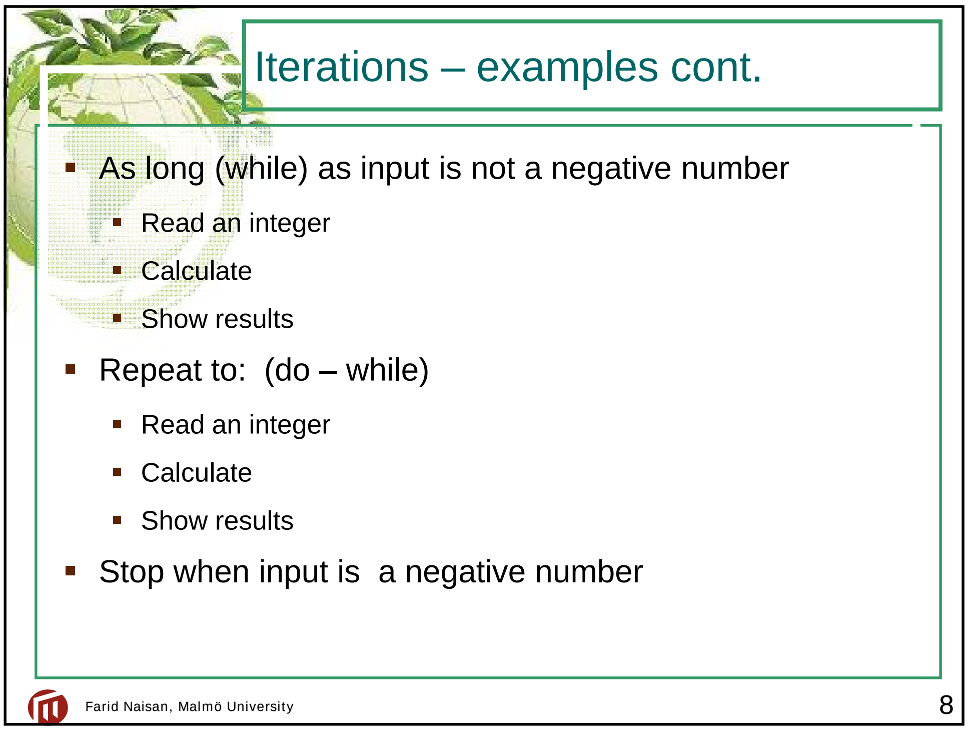
Selection algorithm

- All code executes sequentially. Selection algorithms (if-else) allow you to decide whether a section of code should be executed or not.
- This algorithm is used when making choices between two or more alternatives.
  - if (number >= 0) And (number <= 9)
    - ProceedWithCalculation
  - otherwise
    - Do nothing.



Iteration algorithm

- When certain steps need to be taken repeatedly, loops are used.
- Iterations are performed either for a certain number of times, as long as some condition(s) prevail, or until some condition is met.
  - for year = 2000 to year= 2099
    - if (year / 4) gives a remainder equal to 0
      - the year is a leap year
  - continue with year = year +1

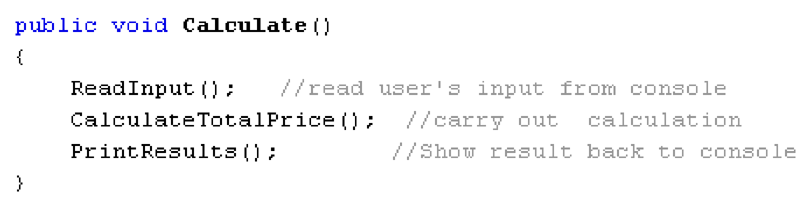


Iterations – examples cont.

- As long (while) as input is not a negative number
  - Read an integer
  - Calculate
  - Show results
- Repeat to: (do – while)
  - Read an integer
  - Calculate
  - Show results
- Stop when input is a negative number

An example of sequential coding

- The figure below shows a sequences of C# statements.



```
public void Calculate()
{
    ReadInput();    //read user's input from console
    CalculateTotalPrice();  //carry out  calculation
    PrintResults();        //Show result back to console
}
```

Operators

- Conditional statements, selection and iteration statements, usually make use of *comparison* and *logical* operators to evaluate Boolean expressions.
- The evaluation of Boolean expressions can only result in a value `true` or `false`.
- There are mainly three groups of operators:
  - Comparison
  - Logical
  - Arithmetic

Comparison evaluation

- Comparison operators compare two values of the same type.
- They cannot be used for values of type String. The operator '==" can however be used due to an explanation that be discussed here.

Operator	Meaning
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
==	is equal to
!=	is not equal to

Comparison operators - example

- The resulting value is either `true` or `false`.

Expression	Meaning	Example: x = 5, y = 7
x > y	Is x greater than y?	`false`
x < y	Is x less than y?	`true`
x >= y	Is x greater than or equal to y?	`false`
x <= y	Is x less than or equal to y.	`true`
x == y	Is x equal to y?	`false`
x != y	Is x not equal to y?	`true`

Logical operators

- Logical operators are used to evaluate two Boolean expressions.
- C# offers two binary (requiring two operands) and one unary (one operand) operators:

the logical AND,	symbol ‘&&’,	binary
the logical OR,	symbol ‘||’,	binary
the logical NOT,	symbol ‘!’,	unary

Logical operators - summary

Operator	Meaning	Description
&&	**AND**	Combines two Boolean expressions into one. Both expressions must be true for the resulting expression to be true.
||	**OR**	Combines two Boolean expressions into one. One or both expressions must be true for the resulting expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	**NOT**	The ! operator reverses the truth of a Boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

The && - operator

- The && operator takes two operands of Boolean type and returns true if and only if both sides are true, and false otherwise.

&& (Logical AND)	
A && B	Result
true && true	**true**
true && false	false
false && true	false
false && false	false

Example: a = 5, b = 46		
Boolean expressions	Result for operands	Result
(a >= 0) && (a <= 9)	**true**&& **true**	**true**
(a >= 0) && (b <= 9)	true && false	false
(a < 0) && (b >= 0)	false && true	false
(a == 0) && (b <= 0)	false && false	false

The || operator

- The logical || operator takes also operands of Boolean type.
- It returns false if and only if both sides are false, and true otherwise.

|| (Logical OR)	
A || B	Result
true || true	true
true || false	true
false || true	true
false || false	**false**

Example: a = 5, b = 46		
Boolean expressions	Result for operands	Result
(a >= 0) || (a <= 9)	true || true	true
(a >= 0) || (b <= 9)	true || false	true
(a < 0) || (b >= 0)	false || true	true
(a == 0) || (b <= 0)	**false** || **false**	**false**

The ! operator

- The ! operator takes one operand and reverses the value from true till false and vice versa.

Logical Expression A	**Result for !A**
true	**false**
false	**true**

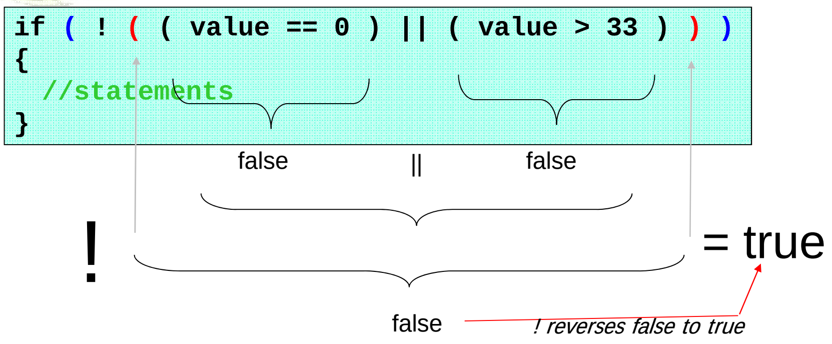
Example: a = 5, b = 46		
Boolean expressions	Result for operand	Result
!(a >= 0)	!true	false
!(b <= 9)	!false	true

The ! operator – cont.

- It is used whenever the true conditions are known but the false condition is of interest.
- *Example: Assume that we (for some reason in a computation) need to skip calculation for values that are 0 or larger than 33. Show the result when the variable **value** (see next slide) = 19 at a certain point during the program execution.*

Logical OR example

- An easy way to detect this situation can be programmed as shown in the figure.
- The result of the whole if-statement is true for value = 19.



```
if ( ! ( ( value == 0 ) || ( value > 33 ) ) )
{
  //statements
}
```

Arithmetic operators

Operator	Meaning	Example
+	Addition	`sum = price + tax;`
-	Subtraction	`cost = price – tax;`
*	Multiplication	`tax = cost * rate;`
/	Division	`salePrice = price / 2;`
%	Modulus	`remainder = sum % 5;`

Example of modulus operator (%)

- Modulus operator gives the remainder after an integer division.
- It is not the whole part but the remainder that becomes the result of the modulus operator.

```
Sum = 15 → 15 % 5 = 0   (15/5 gives rest = 0)
Sum = 16 → 16 % 5 = 1   (16/5 gives rest = 1)
```

- Many times when operations are to carried out for every some value, % operator is useful.

```
if ( year % 4 == 0)  // for every 4th year
   …
```

Short Circuiting

- Logical AND and logical OR operations perform *short-circuit evaluation* of expressions.
- Logical AND will evaluate to false as soon as it sees that one of its operands is a false expression.
- Logical OR will evaluate to true as soon as it sees that one of its operands is a true expression.

Order of Precedence

- The ! operator has a higher order of precedence than the && and || operators.
- The && and || operators have a lower precedence than other logical operators like < and >.
- Parentheses can be used to force the precedence to be changed.

Order of Precedence

Order of Precedence	Operators	Description
1	(unary negation) !	Unary negation, Logical NOT
2	* / %	Multiplication, Division, Modulus
3	+ -	Addition, Subtraction
4	< > <= >=	Less-than, Greater-than, Less-than or equal to, Greater-than or equal to
5	== !=	Is equal to, Is not equal to
6	&&	Logical AND
7	||	Logical OR
8	= += -= *= /= %=	Assignment and combined assignment operators.

The if Statement

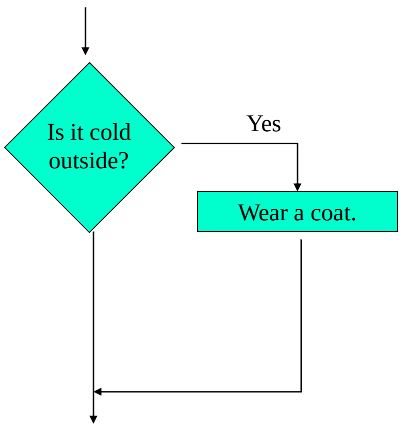
- The code in methods executes sequentially.
- The *if* statement allows the programmer to make decisions on whether a section of code executes or not.
- The if statement uses a Boolean expression as an argument to decide if the next statement or block of statements executes.

```
if (Boolean expression is true)
    execute next statement
```

Flowcharts

- If statements can be modeled as a flow chart.

```
if (coldOutside)
   WearCoat();
```

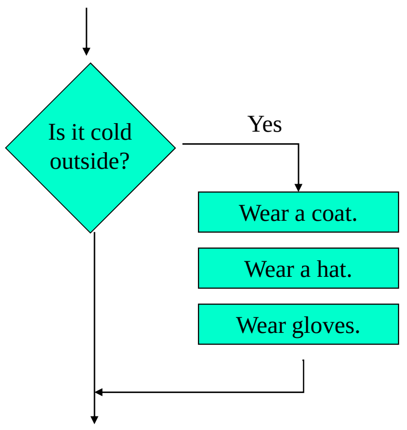


Flowcharts

- A block if statement may be modeled as:

```
if (coldOutside)
{
  WearCoat();
  WearHat();
  WearGloves();
}
```

Note the use of curly braces to block several statements together.



If Statements and Boolean Expressions

```
if (x > y)
    Console.WriteLine(“X is greater than Y”);
if(x == y)
    Console.WriteLine(“X is equal to Y”);
if(x != y)
{
    Console.WriteLine(“X is not equal to Y”);
    x = y;   //assign the value of y to x
    Console.WriteLine(“However, now it is.”);
}
```

Programming Style and if Statements

- If statements can span more than one line; however, they are still one statement.

```
if(average > 95)
   Console.WriteLine("That's a great score!");
```

- is functionally equivalent to

```
if(average > 95) Console.WriteLine("That's a great score!");
```

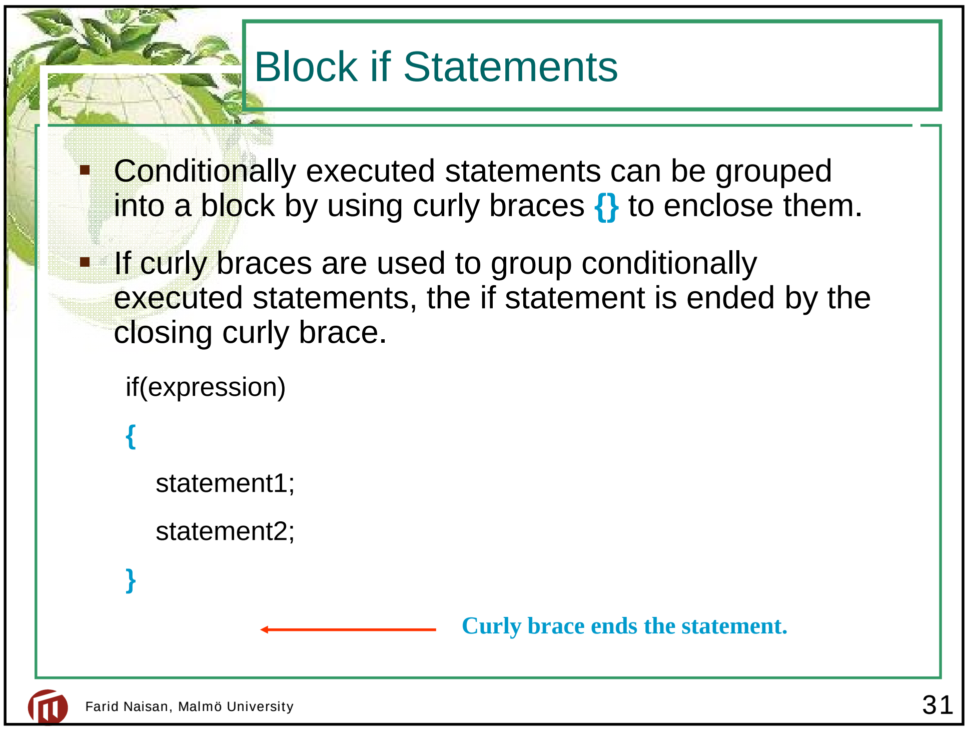
Farid Naisan, Malmö University

Programming Style and if Statements

- Rules of thumb:
  - The conditionally executed statement should be on the line after the if condition.
  - The conditionally executed statement should be indented one level from the if condition.
  - If an if statement does not have the block curly braces, it is ended by the first semicolon encountered after the if condition.

```
if(expression)      <------   No semicolon here.
   statement;     <------     Semicolon ends statement here.
```

Farid Naisan, Malmö University



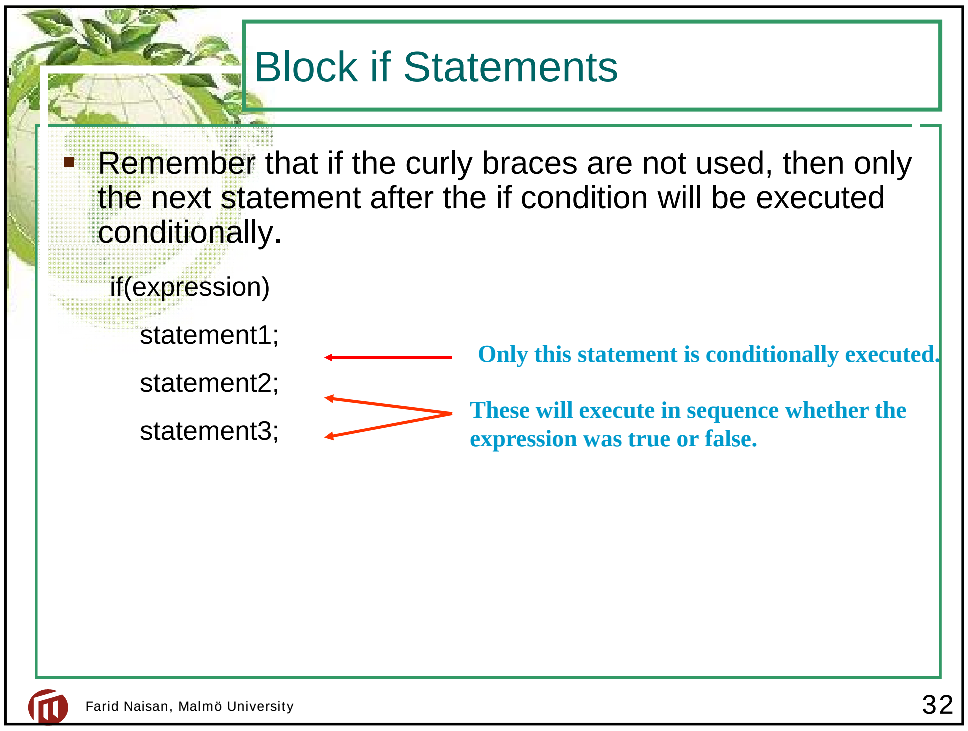
Block if Statements

- Conditionally executed statements can be grouped into a block by using curly braces {} to enclose them.
- If curly braces are used to group conditionally executed statements, the if statement is ended by the closing curly brace.

```
if(expression)
{
    statement1;
    statement2;
}
```

Curly brace ends the statement.

Farid Naisan, Malmö University 31



Block if Statements

- Remember that if the curly braces are not used, then only the next statement after the if condition will be executed conditionally.

```
if(expression)
    statement1;
    statement2;
    statement3;
```

Only this statement is conditionally executed.

These will execute in sequence whether the expression was true or false.

Farid Naisan, Malmö University 32

Flags

- A flag is a Boolean variable that monitors some condition in a program.
- When a condition is true, the flag is set to a true value.

```
if(average > 95)
  highScore = true;
```

- Later, this flag can be tested to determine if the condition has been achieved.

```
if(highScore)
  Console.WriteLine(“That’s a high score!”);
```

Comparing Characters

- Characters can be tested using the relational operators.
- Characters are stored in the computer using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

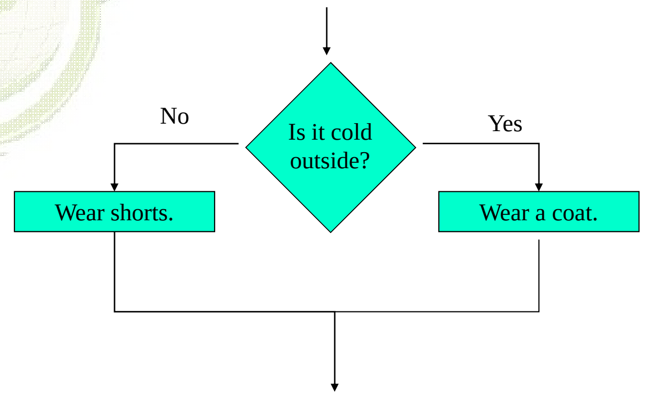
```
char c = ‘A’;
if(c < ‘Z’)
  Console.WriteLine(“A is less than Z);
```

if-else Statements

- The if-else statement adds the ability to conditionally execute code based if the expression of the if statement is false.

```
if(expression)
  statementOrBlockIfTrue;
else
  statementOrBlockIfFalse;
```

if-else Statement Flowcharts



if-else-if Statements

- if-else-if statements can become very complex.
- Imagine the following decision set.

  if it is very cold, wear a heavy coat,

  else, if it is chilly, wear a light jacket,

  else, if it is windy wear a windbreaker,

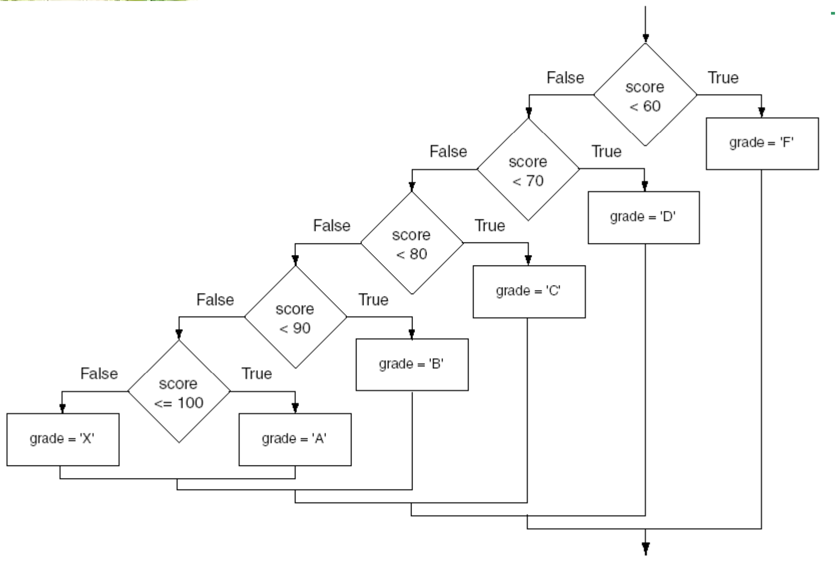
  else, if it is hot, wear no jacket.

if-else-if Statements

```
if (expression)
  statement or block
else if (expression)
  statement or block
  // Put as many else ifs as needed here
else
  statement or block
```

- Care must be used since else statements match up with the immediately preceding unmatched if statement.

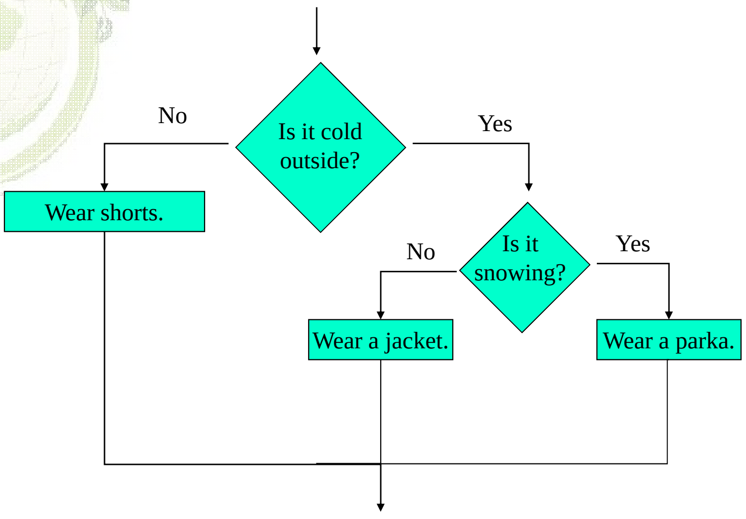
if-else-if Flowchart



Nested if Statements

- If an if statement appears inside of another if statement (single or block) it is called a *nested if* statement.
- The nested if is only executed if the if statement it is in results in a true condition.
- Nested if statements can get very complex, very quickly.
- Too may nested if statements do not look any good; try other algorthims.

Nested if Statement Flowcharts



if-else Matching

- Curly brace use is not required if there is only one statement to be conditionally executed.
- However, sometimes curly braces can help make the program more readable.
- Additionally, proper indentation makes it much easier to match up else statements with their corresponding if statement.

if-else Matching

This else matches with this if.

This else matches with this if.

```
if (salary >= 30000)
{
    if (yearsOnJob >= 2)
    {
       Console.WriteLine("You qualify for the loan.");
    }
    else
    {
       Console.WriteLine("You must have " +
          "been on your current job for at least " +
                 "two years to qualify.");
    }
}
else
{
    Console.WriteLine("You must earn " +
                    "at least $30,000 per year to
  qualify.");
}
```

Comparing string Objects

- In most cases, you cannot use the relational operators to compare two string objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true

Variable Scope

- In C#, a local variable does not have to be declared at the beginning of the method.
- The scope of a local variable begins at the point it is declared and terminates at the end of the method.
- When a program enters a section of code where a variable has scope, that variable has *come into scope*, which means the variable is visible to the program.

The Conditional Operator

- The *conditional operator* is a ternary (three operand) operator.
- The conditional operator allows a programmer a shorthand method of expressing a simple if-else type statement.
- The format of the operators is:

```
expression1 ? expression2 : expression3;
```

- The conditional operator can also return a value.

The Conditional Operator

- The conditional operator can be used as a shortened if-else statement:

```
z = x > y ? 10 : 5;
```

- This line is functionally equivalent to:

```
if(x > y)
  z = 10;
else
  z = 5;
```

The Conditional Operator

- Many times, the conditional operator is used to supply a value.

```
number = x > y ? 10 : 5;
```

- This is functionally equivalent to:

```
if(x > y)
  number = 10;
else
  number = 5;
```

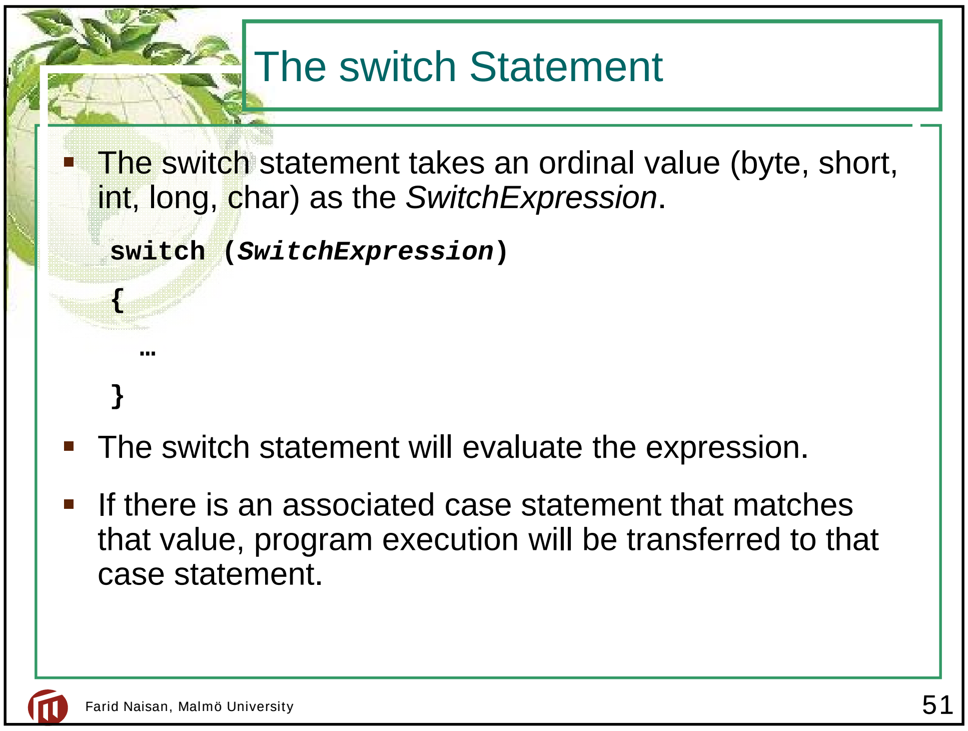
The switch Statement

- The if-else statements allow the programmer to make true / false branches.
- The switch statement allows the programmer to use an ordinal value to determine how a program will branch.
- The switch statement can evaluate an integer type or character type variable and make decisions based on the value.

The switch Statement

- The switch statement takes the form:

```
switch (SwitchExpression)
{
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;
    // case statements may be repeated
    //as many times as necessary
  default:
    // place one or more statements here
    break;
}
```

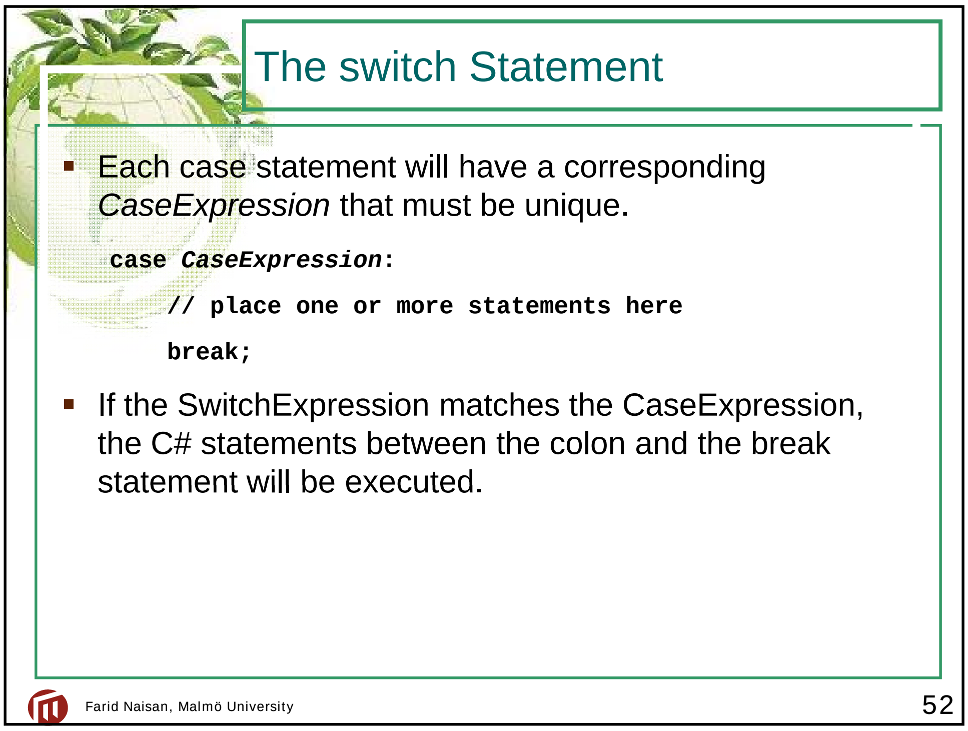


The switch Statement

- The switch statement takes an ordinal value (byte, short, int, long, char) as the *SwitchExpression*.

```
switch (SwitchExpression)
{
  …
}
```

- The switch statement will evaluate the expression.
- If there is an associated case statement that matches that value, program execution will be transferred to that case statement.



The switch Statement

- Each case statement will have a corresponding *CaseExpression* that must be unique.

```
case CaseExpression:
    // place one or more statements here
    break;
```

- If the SwitchExpression matches the CaseExpression, the C# statements between the colon and the break statement will be executed.

The switch Case

- The break statement ends the case statement.
- The break statement is required, otherwise an error will occur.
- The default case is optional and will be executed if no CaseExpression matches the SwitchExpression.
- If you leave it out, your program will have nowhere to branch to if the SwitchExpression doesn't match any of the CaseExpressions.

Summary

- Selection is programmed in C# by using
  - if – else statements
  - switch
- Never use *goto*- statement at any time, any where in your code; it is strictly forbidden in this course!
- The expressions after the keyword if must be inside parenthesis.
- Do not put a semi-colon after the if expression.
- if (x > 0) ; ← Error source