Control structure: Selections

01204111 Computer and Programming

Department of Computer Engineering

Faculty of Engineering
Kasetsart University.

Department of
Computer Engineering
Kasetsart University

MIKE
LABORATORY

Cliparts are taken from *http://openclipart.org*

Outline

- **Boolean Data Type and Expressions**
- Fundamental Control Structures
- Flowcharts: Graphical Representation of Controls
- Basic Selections: if statements
- Basic Selections: if-else statements
- Programming Examples

Boolean Data Type: *bool*

- Type **bool** in C#
- Has two possible values: **true**, **false**
- Declaration Examples of **bool** variables:

```
bool isOdd;
bool isMammal = true;
bool isVaranus = false;
const bool AllHumansAreMortal = true;
```

```
// see how similarly numeric variables are declared
int i = 10;
const double PlanckConstant = 6.6261e-34;
```

Boolean Expressions

- Evaluated to a **bool** value (either **true** or **false**)
- Have two kinds of operators: **relational** and **logical** operators

Relational Operators

==	(equal)
!=	(not equal)
>	(greater than)
<	(less than)
>=	(greater than or equal)
<=	(less than or equal)

Logical Operators

&&	(AND)
||	(OR)
!	(NOT)

Boolean Expressions: *Examples*

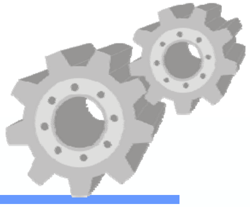
```
int i = 10, j = 15;
bool b, isEven;
b = i < 5;
isEven = (i%2)==0;

Console.WriteLine(b);
Console.WriteLine(isEven);
Console.WriteLine(i >= 5);
Console.WriteLine((i%2)!=0);
Console.WriteLine(!((i%2)==0));
Console.WriteLine(i+j >= 5 && i+j <= 10);
Console.WriteLine(i < 20 || isEven);
```

output

```
False
True
True
False
False
False
True
```

C# Operator Precedence



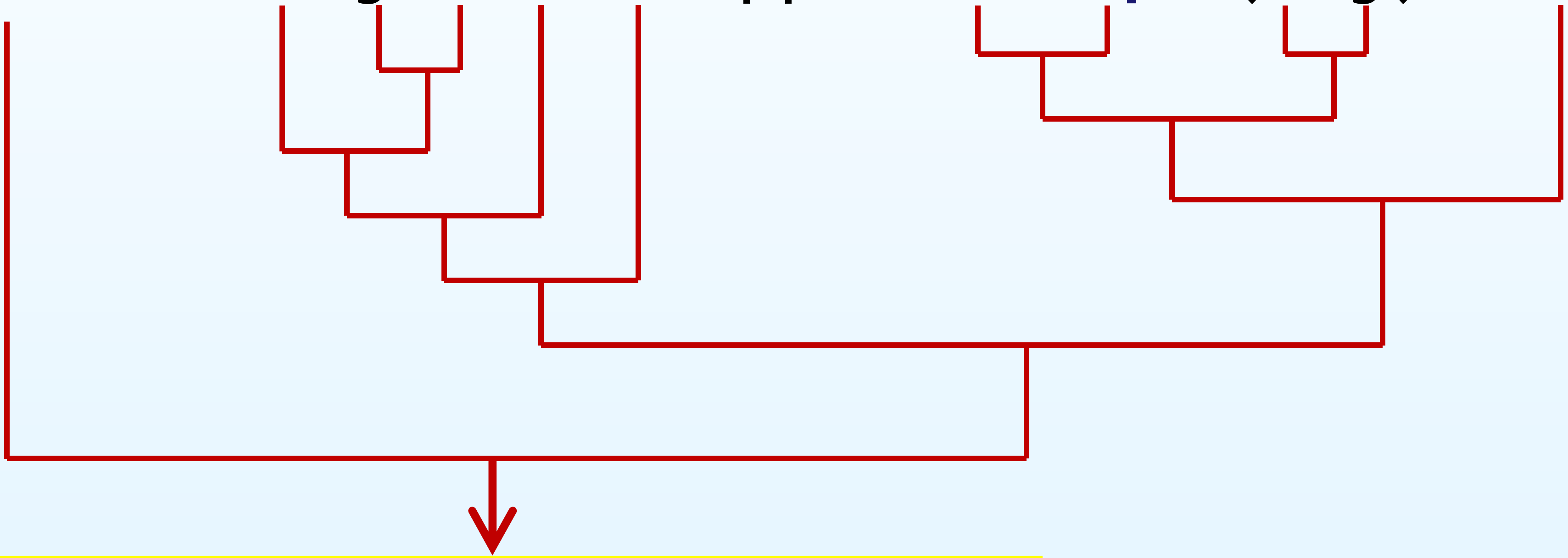
- From the highest precedence to the lowest down the table.
- Operators on the same row have the same precedence.

Category	Operators	Associativity
Primary	(x) x.y f(x) a[x] x++ x--	left to right
Unary	+ - ! ++x --x	left to right
Multiplicative	* / %	left to right
Additive	+ -	left to right
Relational	< > <= >=	left to right
Equality	== !=	left to right
Conditional AND	&&	left to right
Conditional OR	||	left to right
Assignment	= *= /= %= += -=	right to left

Operator Precedence: *Examples*

```
int i = 10, j = 15;
bool passed;

passed = i+j*5-2<10 || Math.Sqrt(i*j)>=20;
```



The result is the value assigned to the variable *passed*

Outline

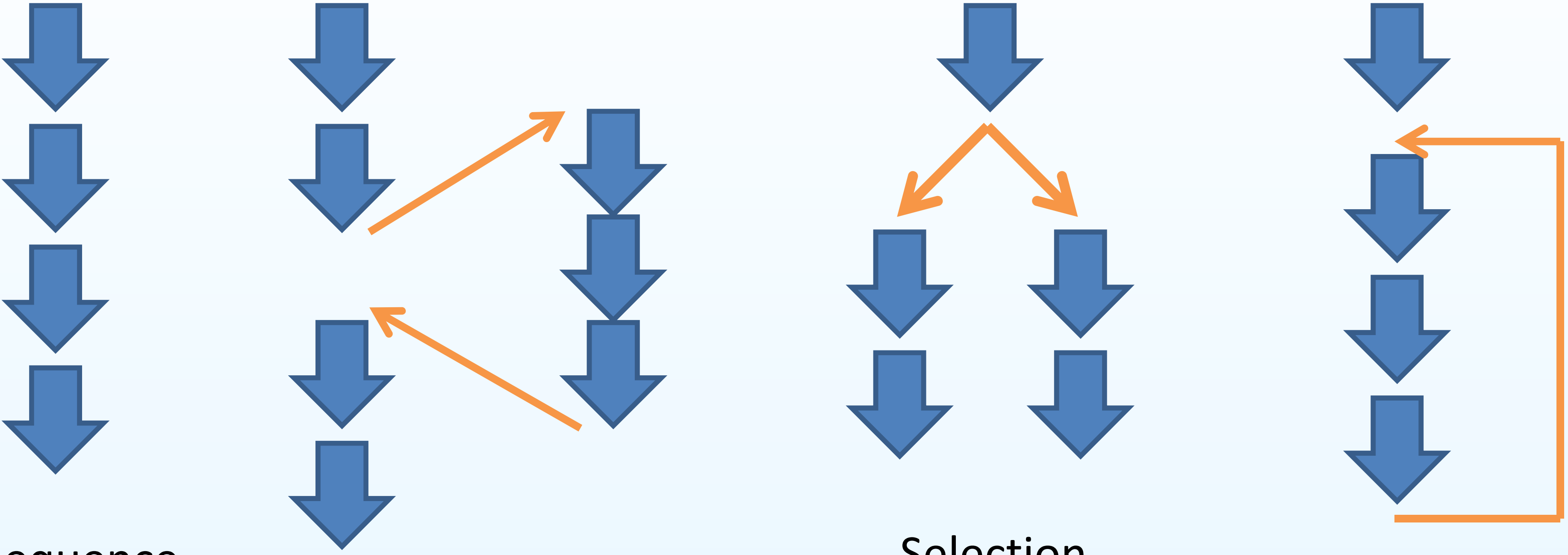
- Boolean Data Type and Expressions
- **Fundamental Control Structures**
- Flowcharts: Graphical Representation of Controls
- Basic Selections: if statements
- Basic Selections: if-else statements
- Programming Examples

Fundamental Control Structures

- **Sequence**
- **Subroutine**
- **Selection** (or **Branching**)
- **Repetition** (or **Iteration** or **Loop**)

You have already learned and used these two control structures.

Fundamental Control Structures



Sequence

Subroutine

Selection

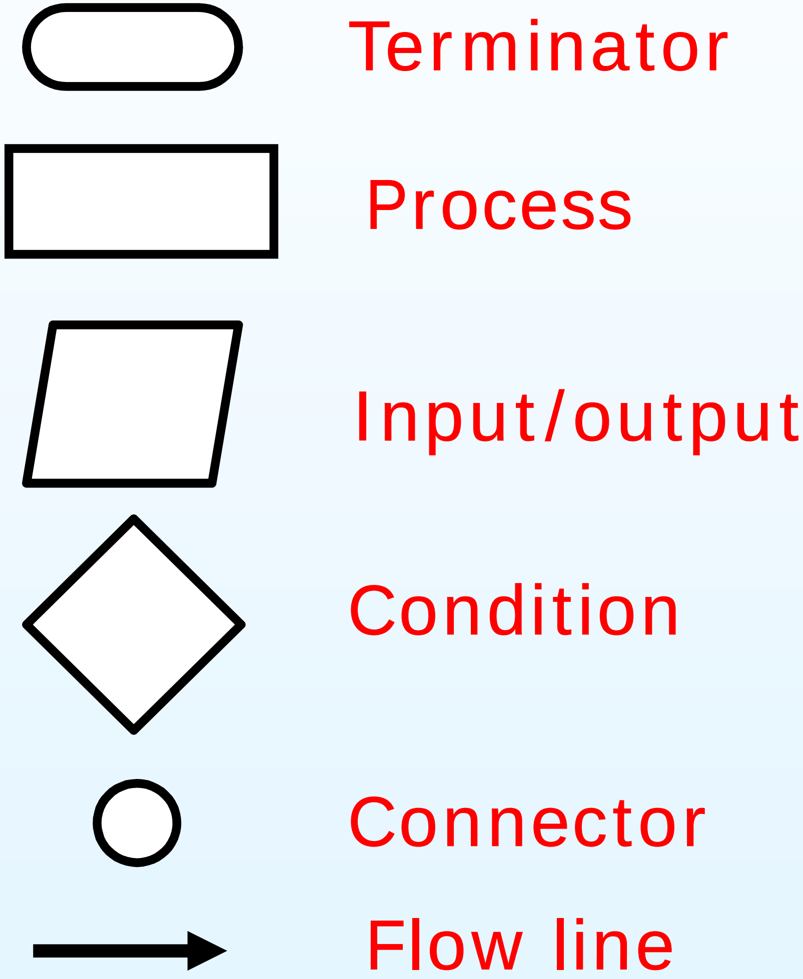
Repetition

Outline

- Boolean Data Type and Expressions
- Fundamental Control Structures
- **Flowcharts: Graphical Representation of Controls**
- Basic Selections: if statements
- Basic Selections: if-else statements
- Programming Examples

Flowcharts: *Graphical Representation of Controls*

Basic flowchart symbols:



Terminator

Process

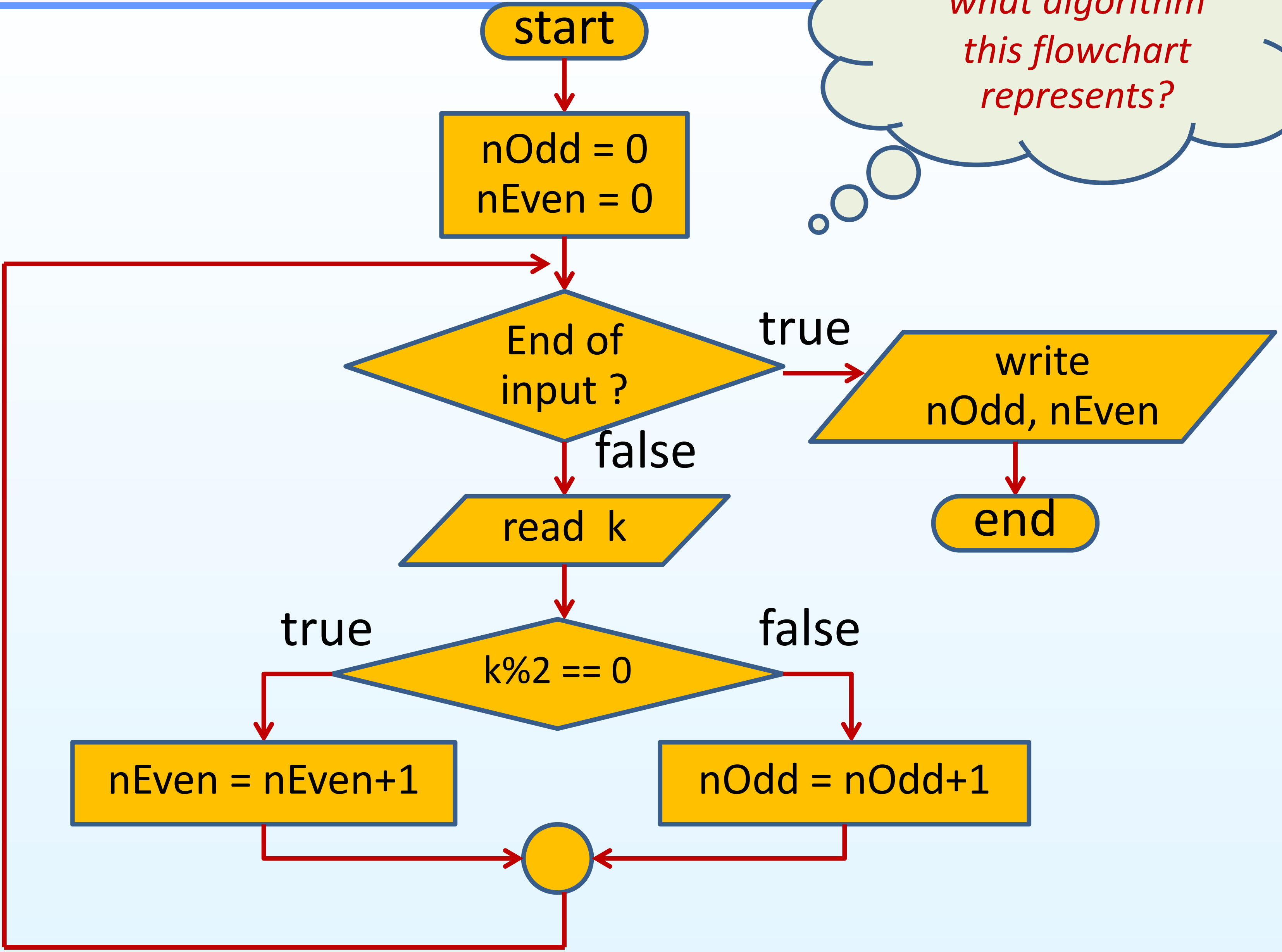
Input/output

Condition

Connector

Flow line

Example:



Can you find out what algorithm this flowchart represents?

Outline

- Boolean Data Type and Expressions
- Fundamental Control Structures
- Flowcharts: Graphical Representation of Controls
- **Basic Selections: if statements**
- Basic Selections: if-else statements
- Programming Examples

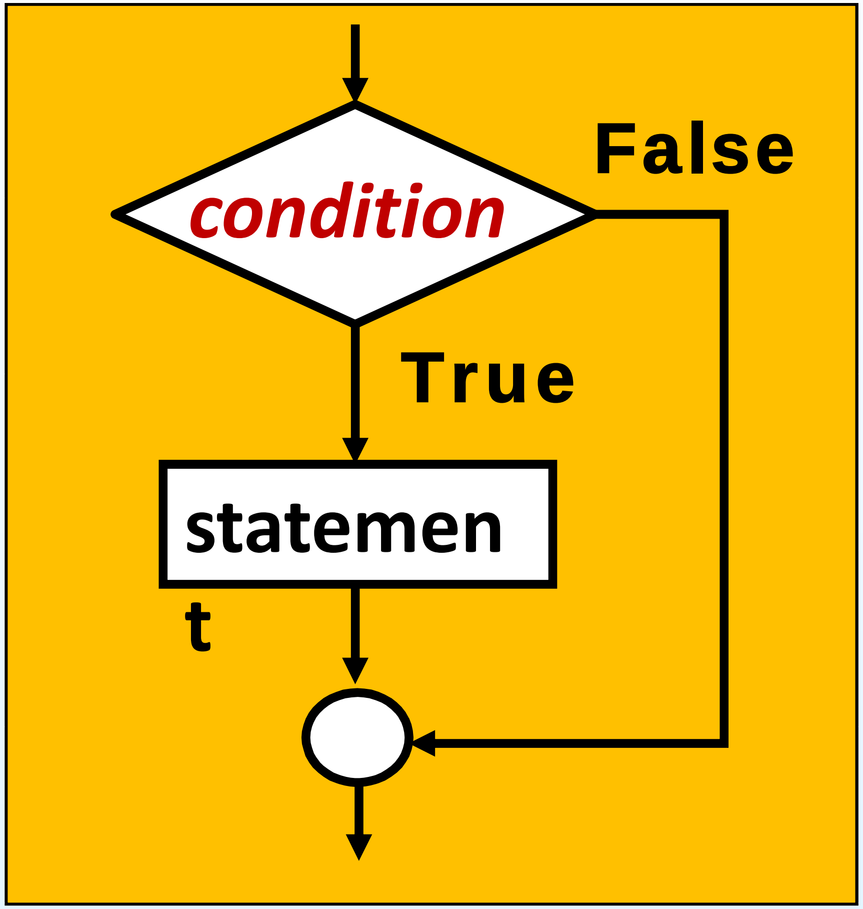
Basic Selection: *if statement*

C# Syntax

```
if (condition)
     statement;
```

- ***Condition*** must be a *boolean expression*

Semantics



Example: *Find the larger of two integers*

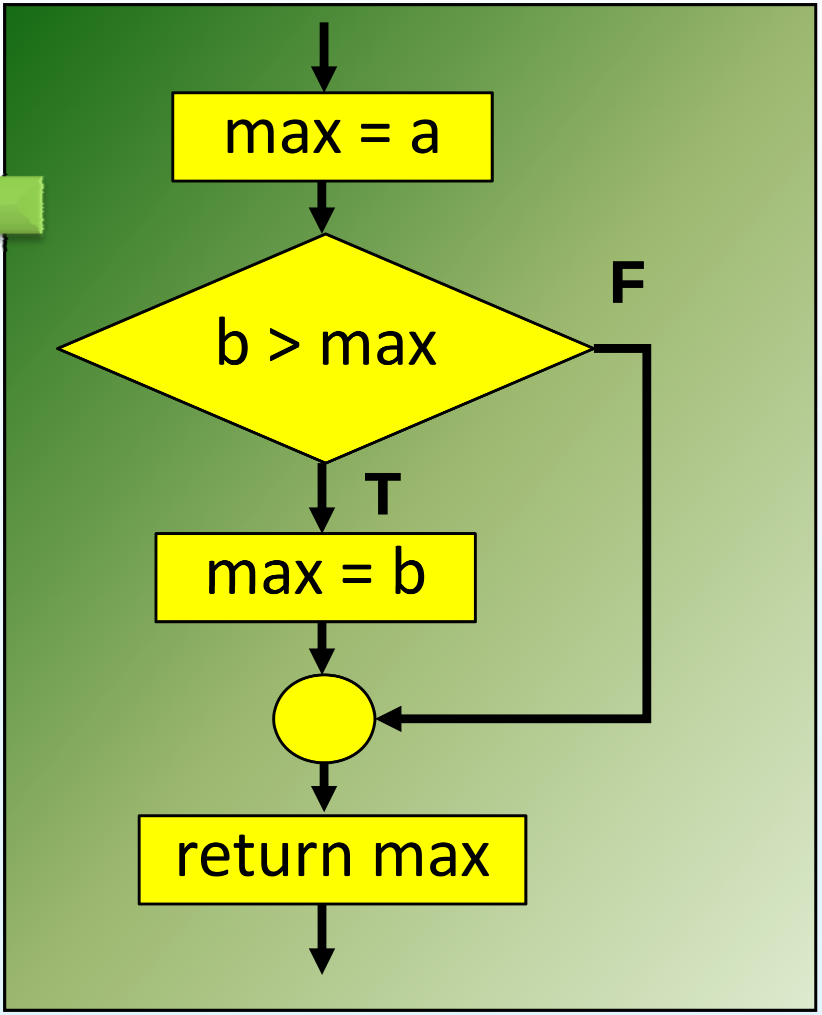
The method ***MaxOfTwo()***

- receives two *int* parameters a and b.
- returns the larger of them.

C# Code

```
static int MaxOfTwo(int a, int b)
{
    int  max = a;
    if (b > max)
        max = b;
    return  max;
}
```

Flow of execution



A Block of Statements

- **A block** is one or more C# statements that are enclosed within a pair of braces {}.
- **A block** is equivalent to *a single C# statement*, so it can be placed wherever a C# statement can be.

```
{
    x = 20;
    y = x+5;
    Console.WriteLine(y);
}
```

A block

```
if (k > 5)
{
    x = 20;
    y = x+5;
    Console.WriteLine(y);
}
```

A block can be placed within an if statement.

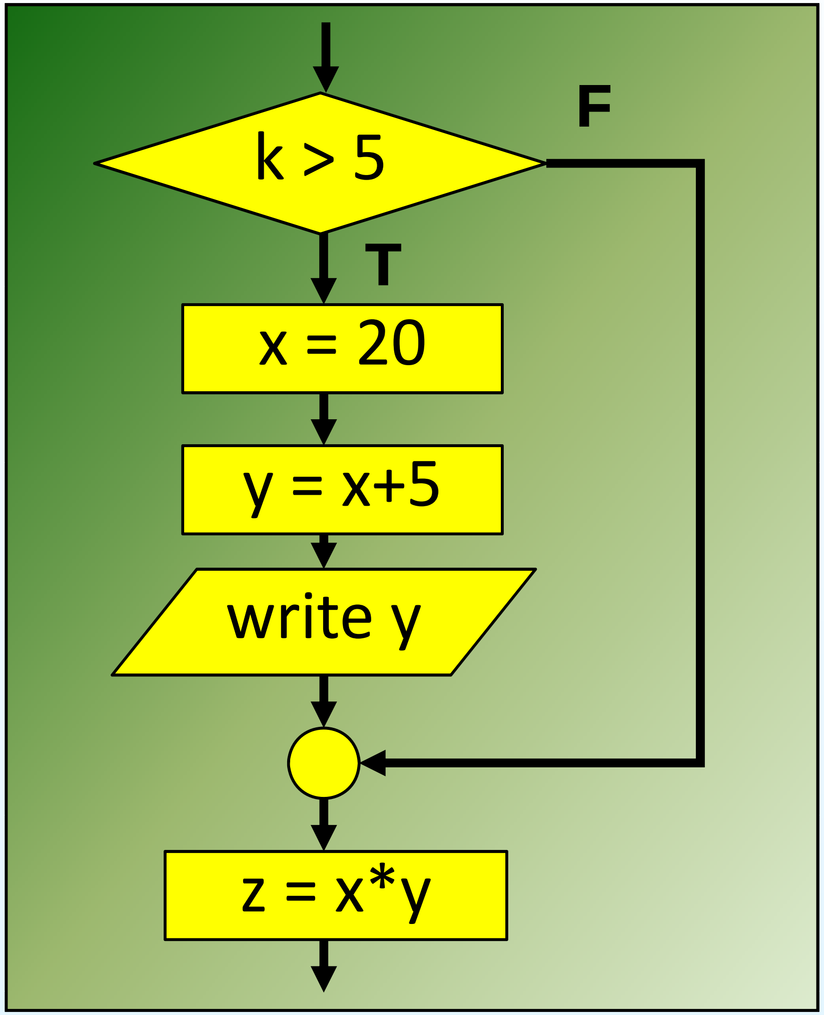
because a block is equivalent to a single statement

Example

C# Code

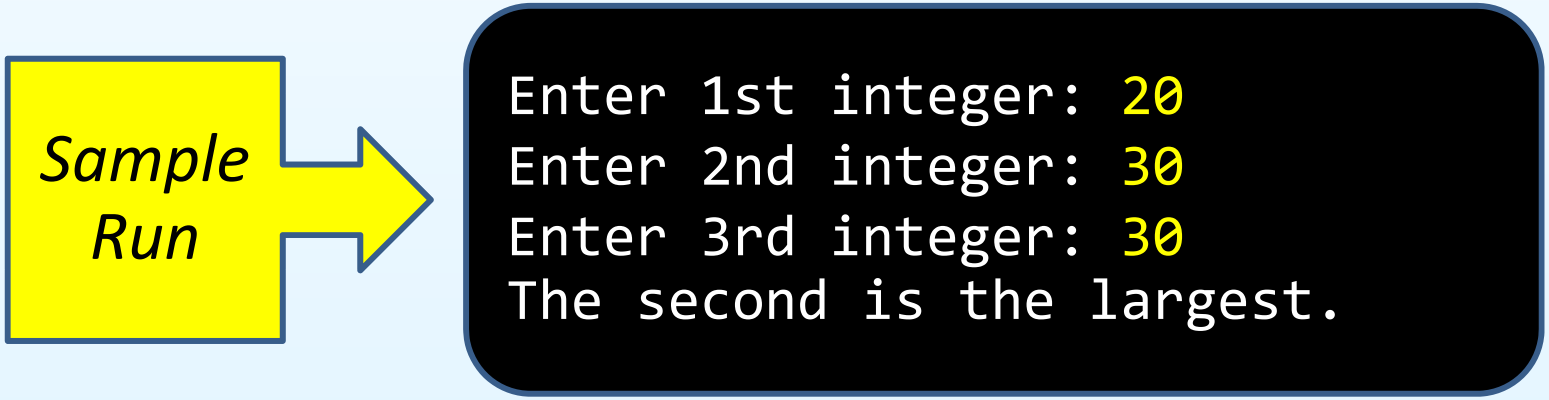
```
if (k > 5)
{
    x = 20;
    y = x+5;
    Console.WriteLine(y);
}
z = x*y;
```

Flow of execution



Task: *The largest of three integers*

- Read three integers
- Find out which of the three is the largest.
- If there are more than one largest numbers, the earlier one wins.



Sample Run

```
Enter 1st integer: 20
Enter 2nd integer: 30
Enter 3rd integer: 30
The second is the largest.
```

19

The largest of three integers – Topmost Level

```
static void Main()
{
    int a = ReadInt("Enter 1st integer: ");
    int b = ReadInt("Enter 2nd integer: ");
    int c = ReadInt("Enter 3rd integer: ");

    Console.WriteLine("The {0} is the largest.",
                      WhichIsLargest(a, b, c));
}
```

To be implemented next

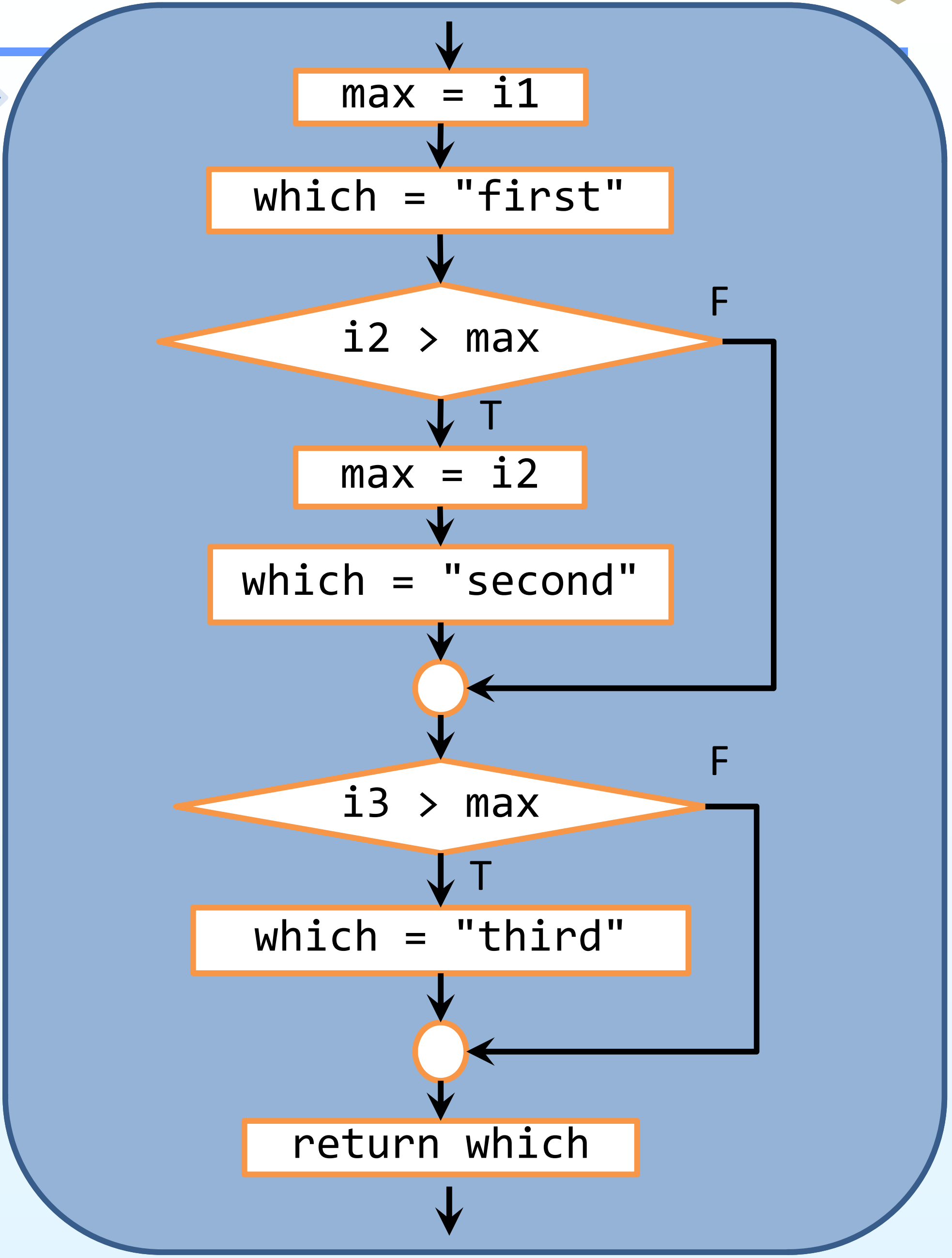
```
static int ReadInt(string prompt)
{
    Console.Write(prompt);
    return int.Parse(Console.ReadLine());
}
```

The Method *WhichIsLargest()* - Steps

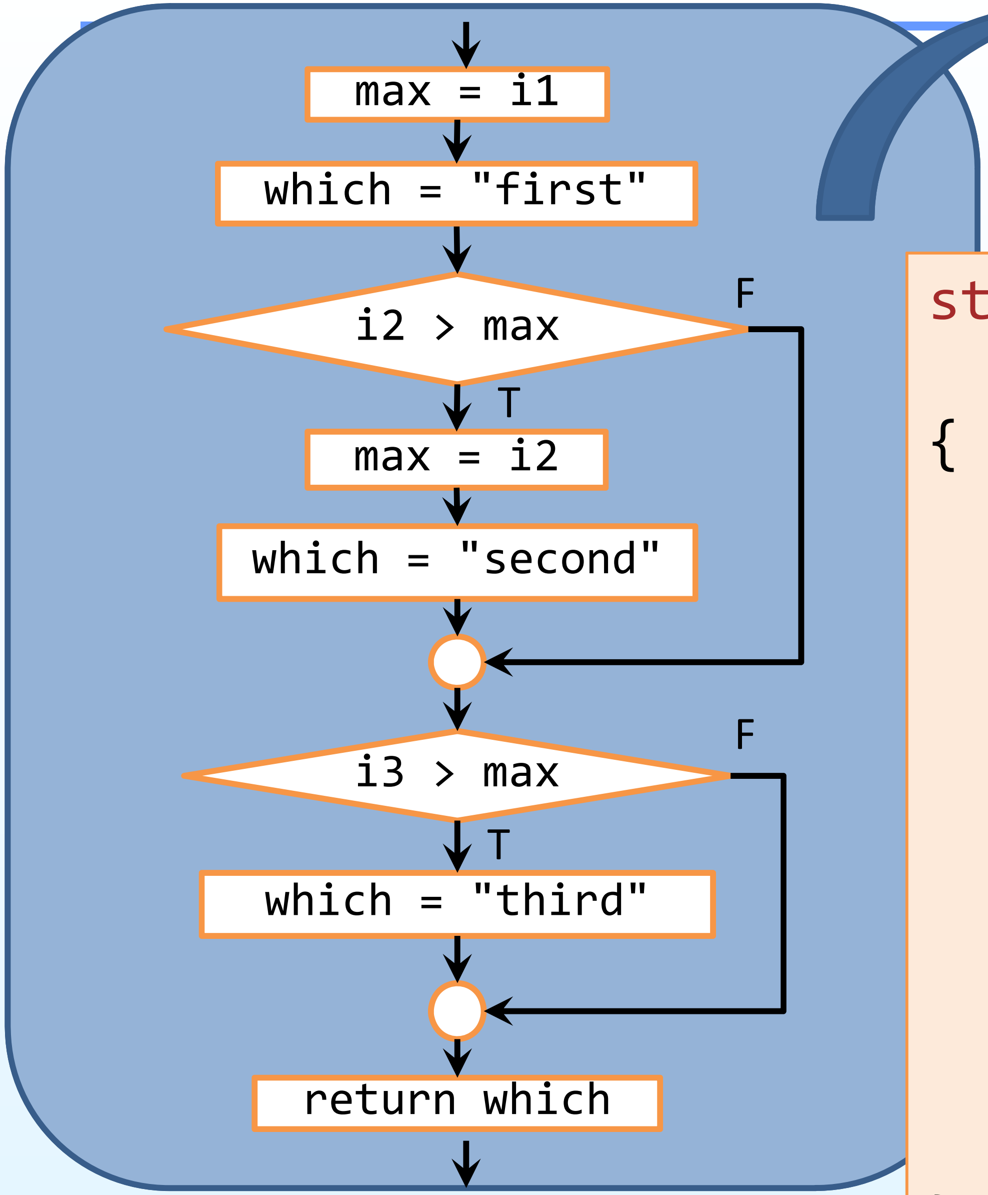
Algorithm

- *Let **i1**, **i2**, and **i3** be the three integers and **max** be the largest so far.*

1. Let max be i1, so the largest so far is the first.
2. If i2 > max, then let max be i2 so the largest so far is the second.
3. If i3 > max, then the largest is the third.



The Method *WhichIsLargest()* – C# code



```
static string WhichIsLargest(
    int i1, int i2, int i3)
{
    int max = i1;
    string which = "first";
    if (i2 > max)
    {
        max = i2;
        which = "second";
    }
    if (i3 > max)
        which = "third";
    return which;
}
```

Outline

- Boolean Data Type and Expressions
- Fundamental Control Structures
- Flowcharts: Graphical Representation of Controls
- Basic Selections: if statements
- **Basic Selections: if-else statements**
- Programming Examples

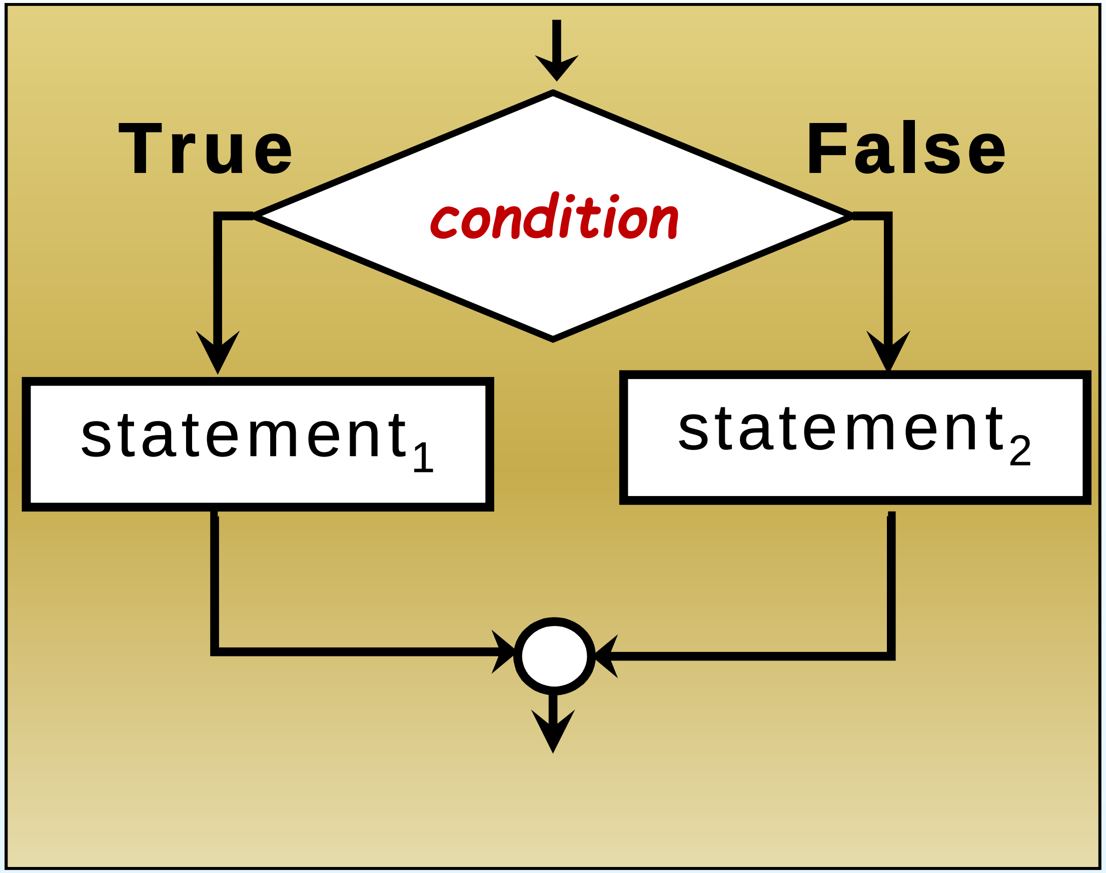
Basic Selection: *if –else statement*

C# Syntax

```
if  (condition)
      statement₁;
else
      statement₂;
```

statement₁ and statement₂ can be a block

Semantics



Example: The method *MaxOfTwo()* revisited

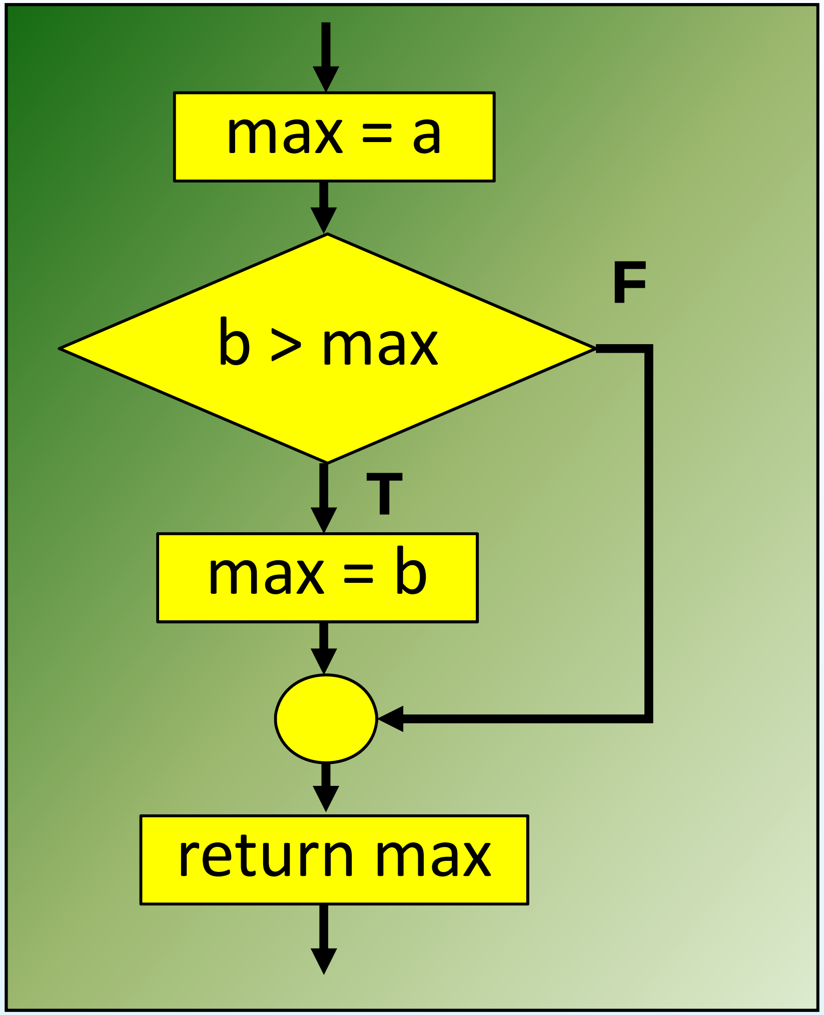
C# Code

```
static int MaxOfTwo(int a, int b)
{
    int  max = a;
    if (b > max)
        max = b;
    return  max;
}
```

This version:

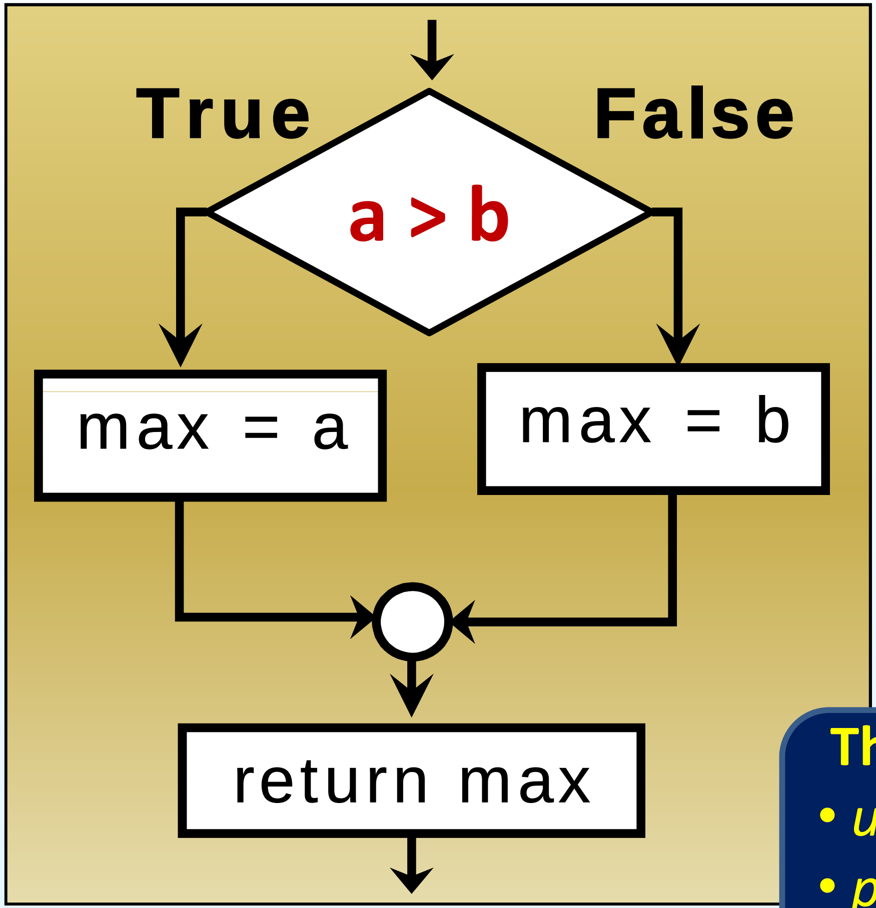
- *uses if (without **else**) statement*
- *performs one comparison*
- *executes one or two assignments*

Flow of execution



Example: *another way to write MaxOfTwo()*

Flow of execution



C# Code

```
static int MaxOfTwo(int a, int b)
{
    int max;
    if (a > b)
        max = a;
    else
        max = b;
    return max;
}
```

This version:

- *uses if-else statement*
- *performs one comparison*
- *always executes only one assignment*

Many styles of placing braces

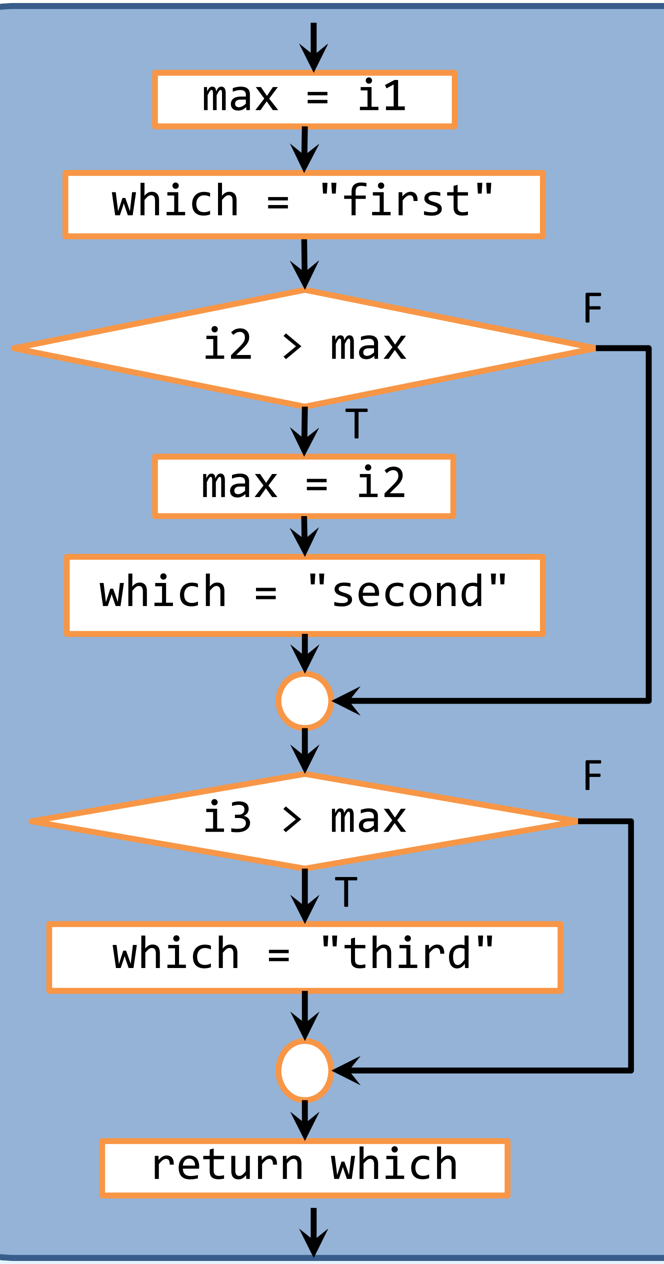
```
if (condition)
{
    Statement_1;
    Statement_2;
}
else
{
    Statement_3;
    Statement_4;
}
Statement_5;
```

```
if (condition)
    {
    Statement_1;
    Statement_2;
    }
else
    {
    Statement_3;
    Statement_4;
    }
Statement_5;
```

```
if (condition) {
    Statement_1;
    Statement_2;
} else {
    Statement_3;
    Statement_4;
}
Statement_5;
```

- To the compiler ***all are the same***, so choose any one you like best.
- Use the same style in the same program.

The Method *WhichIsLargest()* revisited



max = i1

which = "first"

i2 > max

F

T

max = i2

which = "second"

i3 > max

F

T

which = "third"

return which

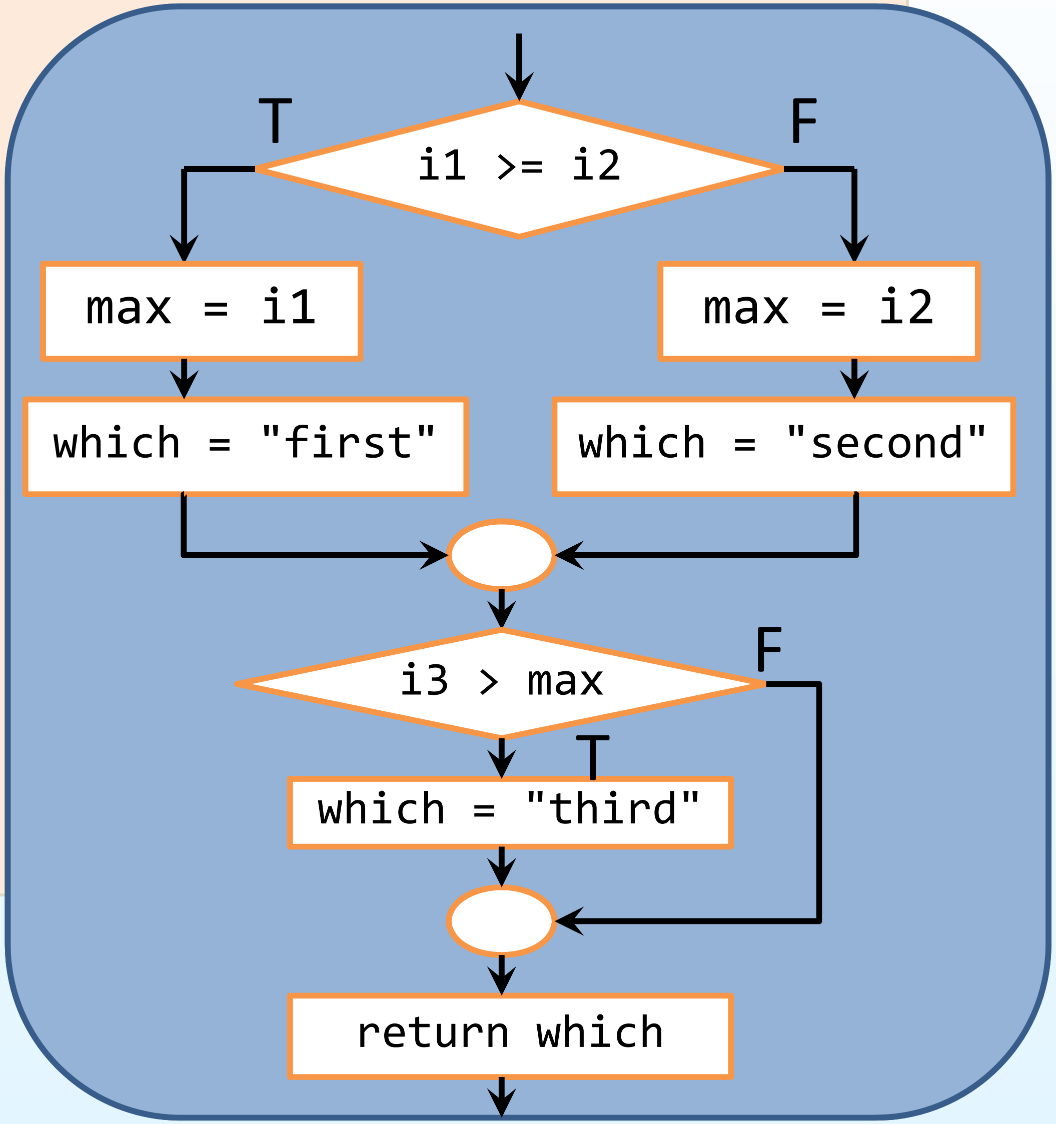
```
static string WhichIsLargest(
    int i1, int i2, int i3)
{
    int max = i1;
    string which = "first";
    if (i2 > max) {
        max = i2;
        which = "second";
    }
    if (i3 > max)
        which = "third";
    return which;
}
```

This version:

- *uses **if** (without **else**) statement*
- *at most 5 assignments are executed*

Example: *another way to write WhichIsLargest ()*

```
// find out which of the three ints is the largest
static string WhichIsLargest(int i1, int i2, int i3)
{
    int max;
    string which;
    if (i1 >= i2) {
        max = i1;
        which = "first";
    } else {
        max = i2;
        which = "second";
    }
    if (i3 > max)
        which = "third";
    return which;
}
```



This version:

- *uses if-else statement*
- *at most 3 assignments are executed*

Outline

- Boolean Data Type and Expressions
- Fundamental Control Structures
- Flowcharts: Graphical Representation of Controls
- Basic Selections: if statements
- Basic Selections: if-else statements
- **Programming Examples**

Task: *Solving quadratic equations*

- Given the three coefficients *a*, *b*, and *c* of a quadratic equation $ax^2 + bx + c = 0$ where $a \neq 0$, find the *roots* of the equation.

A ***root*** is a value of x that satisfies the equation

Solving quadratic equations - I/O Specification

Sample Run

```
Enter 1st coefficient: 0
1st coefficient can’t be zero. Program exits.
```

Sample Run

```
Enter 1st coefficient: 1
Enter 2nd coefficient: 8
Enter 3rd coefficient: 16
Only one real root: -4
```

Sample Run

```
Enter 1st coefficient: 2
Enter 2nd coefficient: -1
Enter 3rd coefficient: -1
Two real roots: 1 and -0.5
```

Sample Run

```
Enter 1st coefficient: 5
Enter 2nd coefficient: 2
Enter 3rd coefficient: 1
Two complex roots: -0.2+0.4i and -0.2-0.4i
```

Solving quadratic equations - Ideas

- The *roots* of a quadratic equation *$ax^2 + bx + c = 0$* can be calculated by the formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- The term ***$b^2 - 4ac$*** in the formula is called the **discriminant** (D) of the equation because it can discriminate between the possible types of roots.

Solving quadratic equations - Ideas

The discriminant *$D = b^2 - 4ac$* of the equation determines the type of roots as follows:

- *If D > 0, there are two real roots:* $\frac{-b+\sqrt{D}}{2a}$ *and* $\frac{-b-\sqrt{D}}{2a}$
- *If D = 0, there is only one real root:* $\frac{-b}{2a}$
- *If D < 0, there are two complex roots:*

$$\frac{-b}{2a}+i\frac{\sqrt{-D}}{2a} \quad \text{and} \quad \frac{-b}{2a}-i\frac{\sqrt{-D}}{2a}$$

Now we have got enough information to write the program.

Solving a quadratic equation – Topmost Steps

❖ The Main() method:

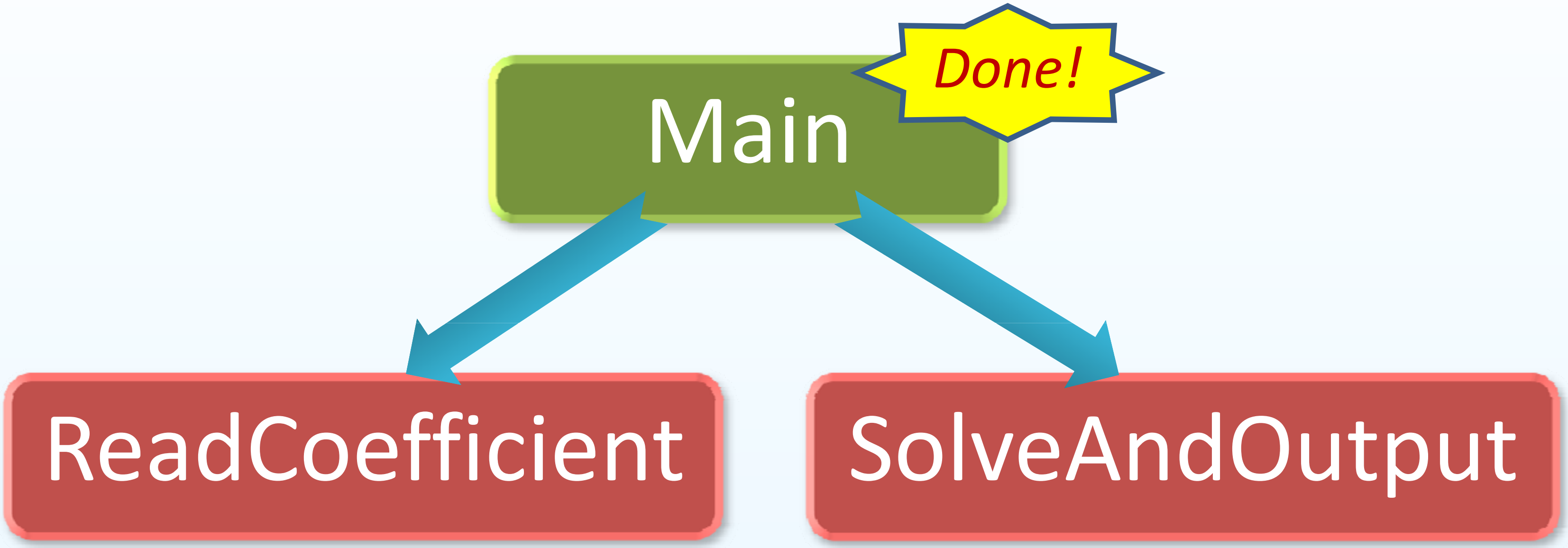
1. reads the three coefficients *a, b,* and *c*
2. uses *a, b,* and *c* to solve and output the roots.

```
public static void Main()
{
        double a, b, c;
        ReadCoefficients(out a, out b, out c);
        SolveAndOutput(a, b, c);

        Console.ReadKey(true);
}
```

The supreme commander usually doesn’t do things himself. He only gives orders.

Solving a quadratic equation – Call Tree



Solving a quadratic equation – Read the inputs

❖ The method *ReadCoefficient()*

1. reads the coefficients *a, b,* and *c* by calling *ReadDouble()* for each.
2. If *a* is zero, it prints an error message and then terminates the program immediately.

What command could we use to terminate a running program immediately?

For Console applications, we could use the System method ***Environment.Exit()*** with an ***exit code***.

A System method: *Environment.Exit()*

- **Namespace:** `System`
- **Class:** `Environment`
- **Method:**

```
public static void Exit(int exitCode)
```

- **Description:**

  Exit() terminates this running program and returns an exit code of type *int* to the operating system.

- **Parameter:**
  - ***exitCode*** is an integer of type *int* to return to the operating system.
  - Traditionally, **zero** is used to indicate a *successful exit* and a **nonzero number** is used to indicate *an error*.
  - You can define your own error codes by various nonzero numbers.

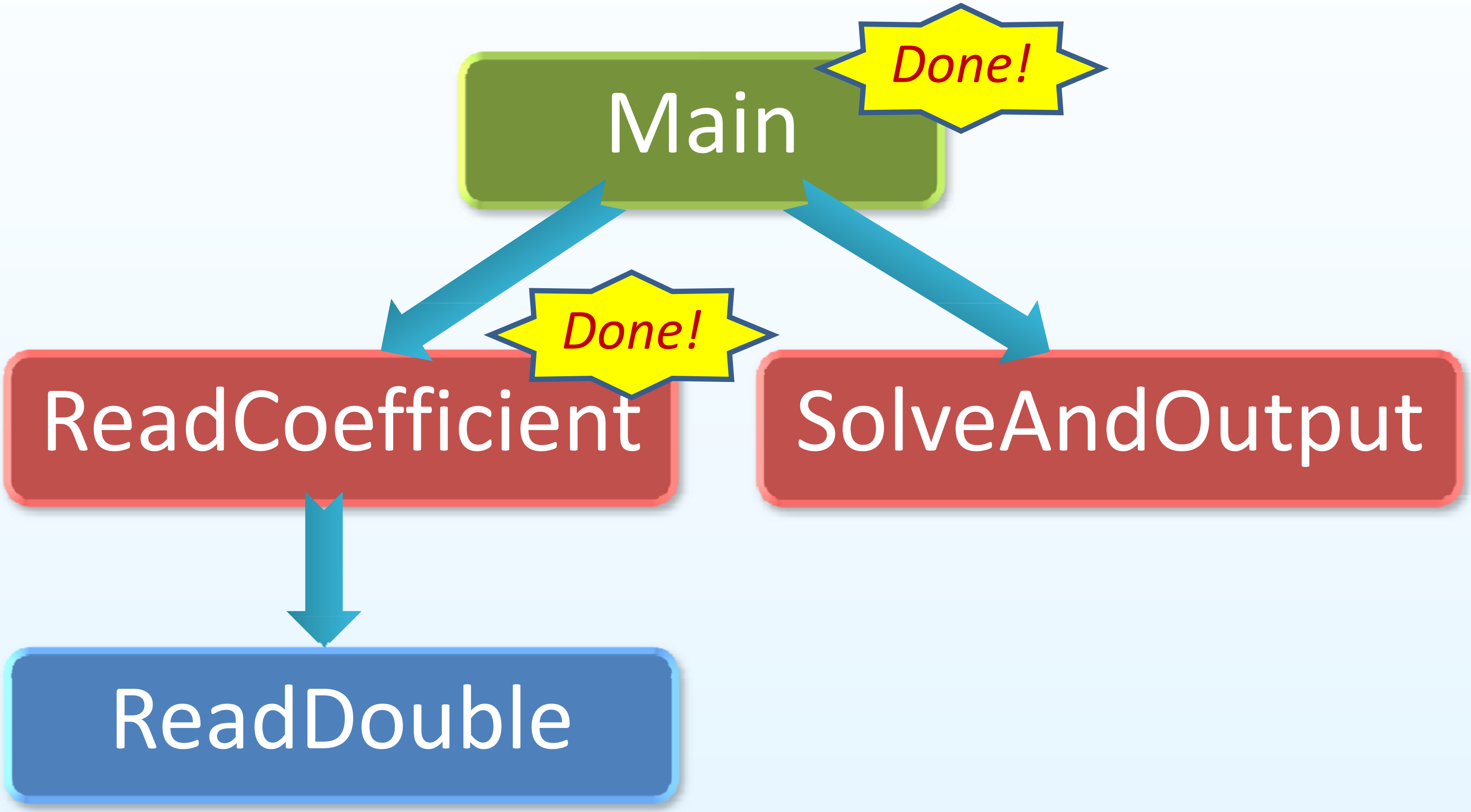
Solving a quadratic equation – Read the inputs

- The method *ReadCoefficient()*

1. reads the coefficients *a, b,* and *c* by calling *ReadDouble()*.
2. If *a* is zero, it prints an error message and then

```
static void ReadCoefficients(out double a, out double b, out double c)
{
  a = ReadDouble("Enter 1st coefficient: ");
  if (a == 0) {
     Console.WriteLine("1st coefficient can't be zero. Program exits.");
     Console.ReadKey(true);
     Environment.Exit(1);    // nonzero exit code to indicate error
  }
  b = ReadDouble("Enter 2nd coefficient: ");
  c = ReadDouble("Enter 3rd coefficient: ");
}
```

Solving a quadratic equation – Call Tree



Main

Done!

ReadCoefficient

Done!

SolveAndOutput

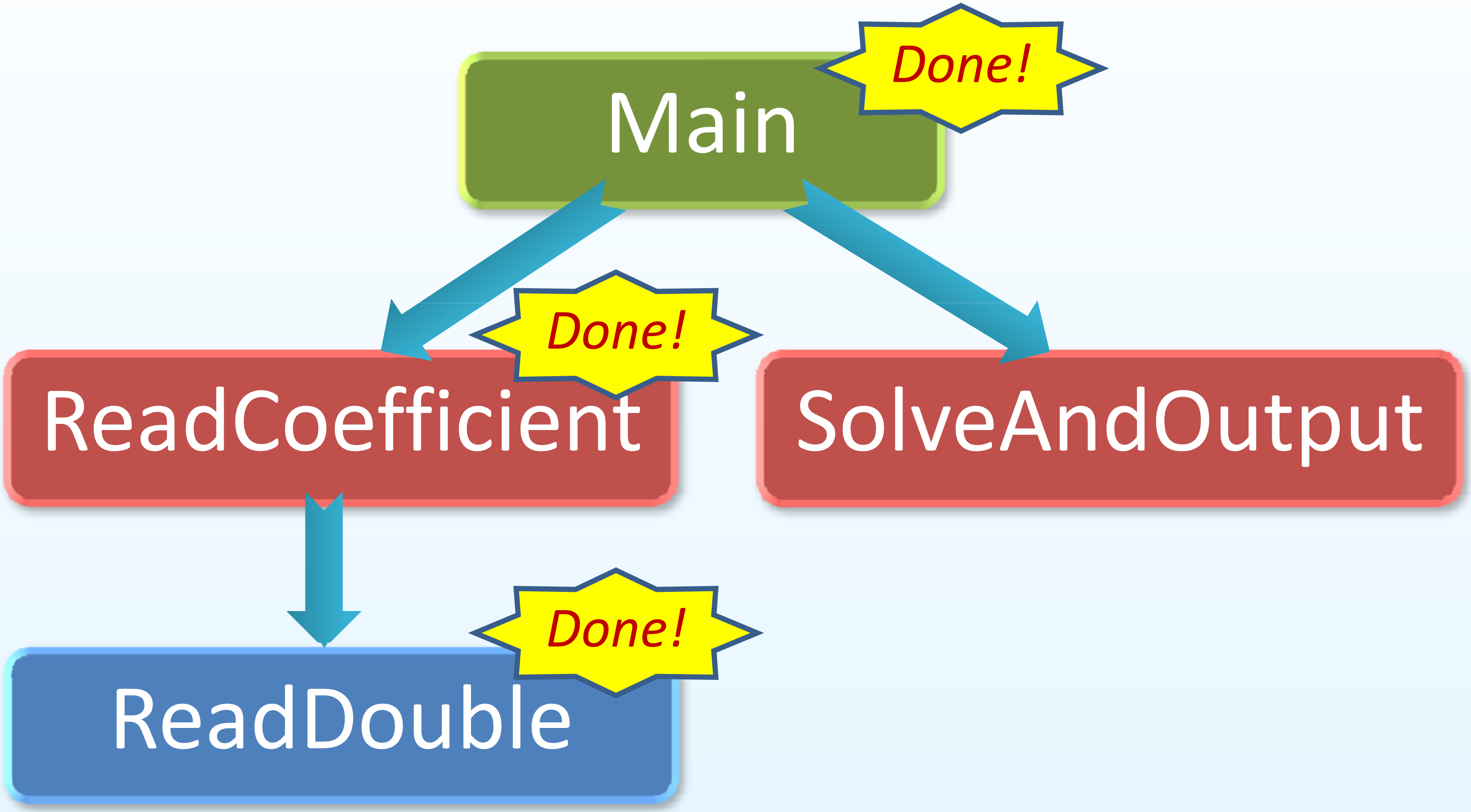
ReadDouble

Solving a quadratic equation - Program

❖Our good old method *ReadDouble()*

```
static double ReadDouble(string prompt)
{
        Console.Write(prompt);
        return double.Parse(Console.ReadLine());
}
```

Solving a quadratic equation – Call Tree



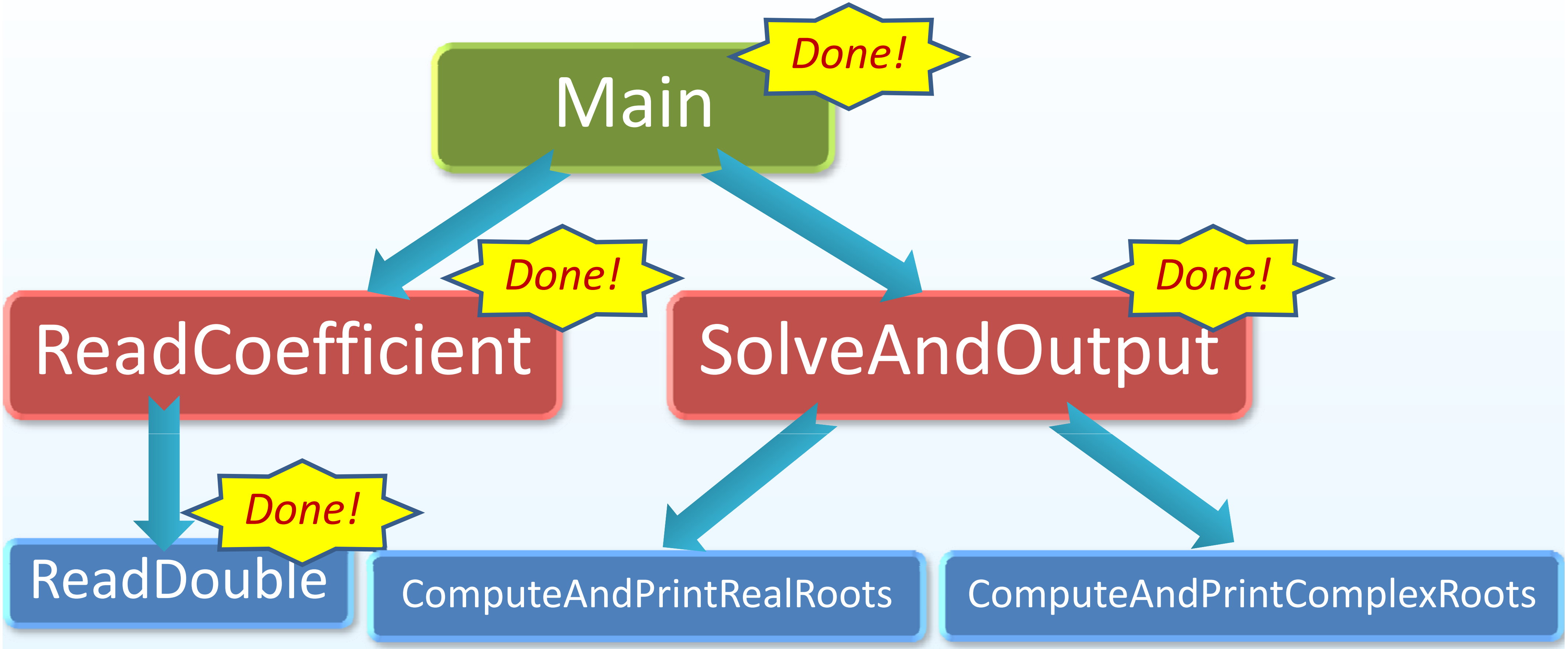
Solving a quadratic equation - Prograr

- The method *SolveAndOutput()*
  1. computes the *discriminant.*
  2. uses the *discriminant* to select either the method to find ***real*** roots or the one to find

```
static void SolveAndOutput(double a, double b, double c)
{
      double discriminant = b*b - 4*a*c;

      if (discriminant >= 0)     // has real roots
           ComputeAndPrintRealRoots(a, b, c);
      else                       // has complex roots
           ComputeAndPrintComplexRoots(a, b, c);
}
```

Solving a quadratic equation – Call Tree



Solving quadratic equations - Ideas

Before we go further, let's recall the formula

The discriminant $D = b^2 - 4ac$ of the equation determines the type of roots as follows:

- *If D > 0, there are two real roots:*

$$\frac{-b+\sqrt{D}}{2a} \quad \text{and} \quad \frac{-b-\sqrt{D}}{2a}$$

- *If D = 0, there is only one real root:* $\frac{-b}{2a}$

- *If D < 0, there are two complex roots:*

$$\frac{-b}{2a}+i\frac{\sqrt{-D}}{2a} \quad \text{and} \quad \frac{-b}{2a}-i\frac{\sqrt{-D}}{2a}$$

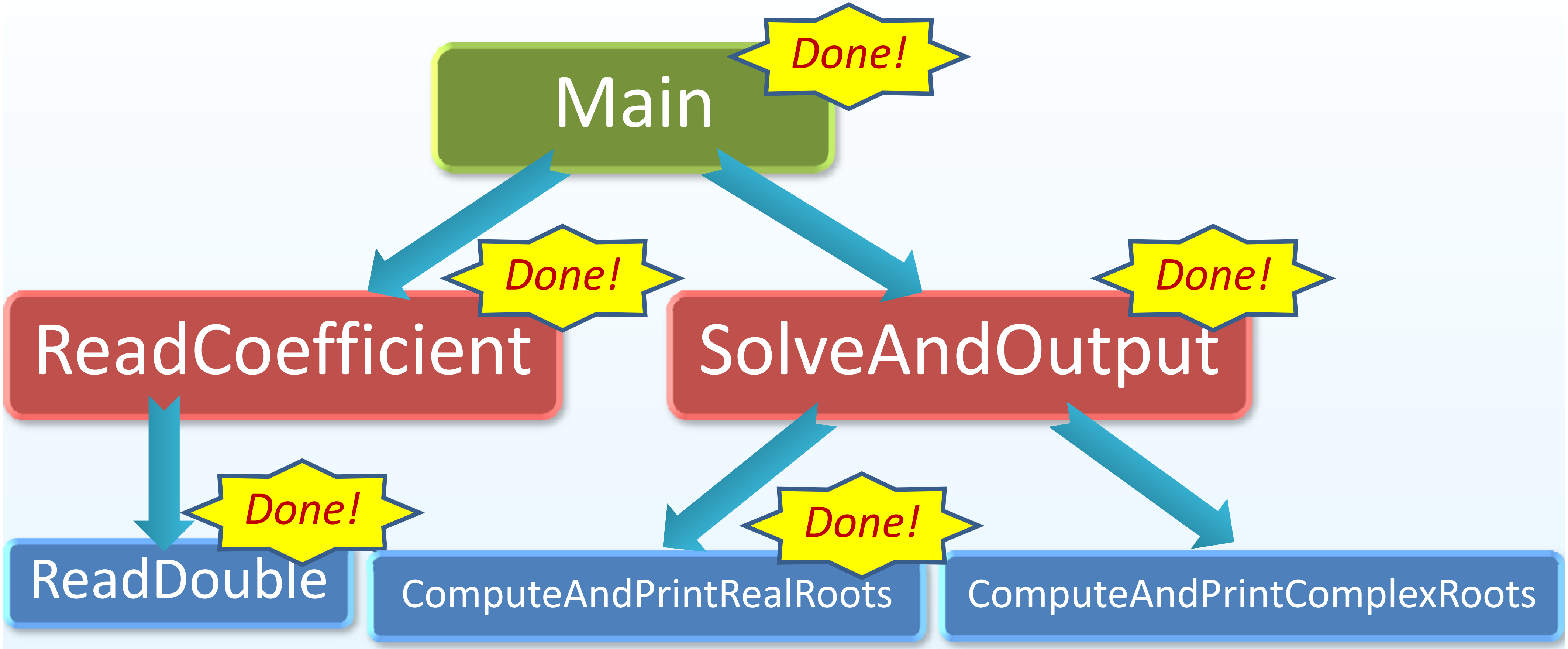
Solving a quadratic equation - Program

❖The method *ComputeAndPrintRealRoots()*

1. uses the *discriminant* to select either the formula for ***one*** real root or ***two*** real roots.
2. computes and outputs the root(s).

```
static void ComputeAndPrintRealRoots(double a, double b, double c)
{
     double r1, r2;
     double discrim = b*b - 4*a*c;
     if (discrim == 0) {
          r1 = -b / (2*a);
          Console.WriteLine("Only one real root: {0}", r1);
     } else {
          r1 = (-b + Math.Sqrt(discrim)) / (2*a);
          r2 = (-b - Math.Sqrt(discrim)) / (2*a);
          Console.WriteLine("Two real roots: {0} and {1}", r1, r2)
     }
}
```

Solving a quadratic equation – Call Tree



Main

Done!

ReadCoefficient

Done!

SolveAndOutput

Done!

ReadDouble

Done!

ComputeAndPrintRealRoots

Done!

ComputeAndPrintComplexRoots

Solving a quadratic equation - Program

- The method *ComputeAndPrintComplexRoots()*

```
static void ComputeAndPrintComplexRoots(double a, double b, double c)
{



}
```

Now it's time
for all good students
to write it yourself!

Conclusion

- **Control structures** allow you to control the flow of your program's execution
- There are four fundamental control structures: *Sequence*, *Subroutine*, *Selection*, and *Repetition*. The previous chapters have already used the first two.
- The control structure *Selection* is used to select one of many possible paths of execution in a program depending on the given conditions. Each condition is expressed in C# by a *bool* expression.
- In C#, *Selection* can be expressed by the *if* statements or *if-else* statements. The *if* statement decides whether or not a statement (or a block) is to be executed. The *if-else* statement selects between two possible statements (or blocks) to be executed.

References

- Data type *bool* and *bool* expressions:
  https://msdn.microsoft.com/en-us/library/c8f5xwh7.aspx
- C# operators *(from the highest precedence to the lowest)*
  https://msdn.microsoft.com/en-us/library/6a71f45d.aspx
- *if* and *if-else* statements:
  https://msdn.microsoft.com/en-us/library/5011f09h.aspx
- A block of statements:
  https://msdn.microsoft.com/en-us/library/ms173143.aspx
- *Environment.Exit()* method:
  https://msdn.microsoft.com/en-us/library/system.environment.exit.aspx

Syntax Summary I

if statement

```
if (condition)
    statement;
```

if-else statement

```
if (condition)
    statement_1;
else
    statement_2;
```

Condition must be a ***bool*** expression.

A Block

```
{
    statement_1;
    statement_2;
    ...
    statement_k;
}
```

A block can be anywhere a single statement can be.

A System Method

```
Environment.Exit(exitCode);
```

terminates the program immediately and returns *exitCode* to the OS.

Syntax Summary II: *C# Operator Precedence*

- From the highest precedence to the lowest down the table.
- Operators on the same row have the same precedence.

Category	Operators	Associativity
Primary	`(x)   x.y   f(x)   a[x]   x++   x--`	left to right
Unary	`+   -   !   ++x   --x`	left to right
Multiplicative	`* / %`	left to right
Additive	`+   -`	left to right
Relational	`<   >   <=   >=`	left to right
Equality	`==   !=`	left to right
Conditional AND	&&	left to right
Conditional OR	||	left to right
Assignment	`=   *=   /=   %=   +=   -=`	right to left