



# COMPSCI 280 S1 2010 Enterprise Software Development

Programming Fundamentals  
Control Flow

Slides in this section are largely based on Angela Chang's lecture notes with minor modification – thanks Angela!



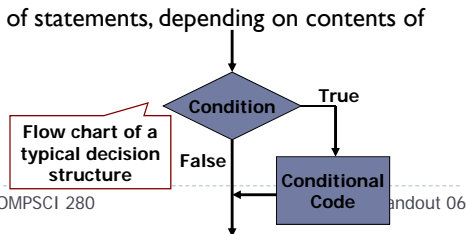
## Agenda & Reading

- ▶ **Agenda:**
  - ▶ Decision-making statements:
    - ▶ if, if-else, nested if-else, if-else-if, switch
  - ▶ Loops
    - ▶ while, do-While, for,
  - ▶ Nested Loops
  - ▶ Others
    - ▶ break, continue, return
    - ▶ using
- ▶ **Recommended Reading:**
  - ▶ Statements (C# Programming Guide)
    - ▶ <http://msdn2.microsoft.com/en-us/library/ms173143.aspx>
  - ▶ C# for Java Programmers
    - ▶ Chapter 4
  - ▶ Microsoft Visual C# 2008 Step by Step
    - ▶ Chapter 4 - 5
- ▶ **Hands-On Lab:**
  - ▶ Lecture06Lab

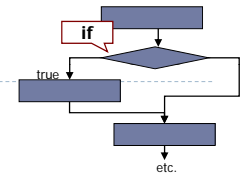


## Decision-Making Statements

- ▶ Decision-Making statements evaluate conditions and execute statements based on that evaluation
- ▶ VB .NET includes two decision-making statements:
  - ▶ If statement
    - ▶ Evaluates an expression
    - ▶ Executes one or more statements if expression is true
    - ▶ Can execute another statement or group of statements if expression is false
  - ▶ Switch statement
    - ▶ Evaluates a variable for multiple values
    - ▶ Executes a statement or group of statements, depending on contents of the variable being evaluated



## if & if-else



- ▶ The if statement
  - ▶ The block governed by it is executed if a condition is true
  - ▶ The Boolean\_Expression must be enclosed in parentheses
  - ▶ The statement\_when\_true branch of an if can be a made up of a single statement or a compound statement
  - ▶ Note:
    - ▶ A compound statement is made up of a list of statements and must always be enclosed in a pair of braces ({})

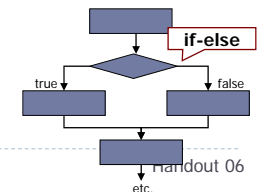
```
if (boolean_expression)
  statement_when_true;
```

```
if (boolean_expression) {
  statements_when_true;
  ...
}
```

- ▶ The If-else statement
  - ▶ It is two-way selection.
  - ▶ The else block is executed if the if part is false.
  - ▶ Again, the statements\_when\_true or statements\_when\_false can be a made of a single statement or many statements.

```
if (boolean_expression)
  statement_when_true;
else
  statement_when_false;
```

```
if (boolean_expression){
  statements_when_true;
  ...
} else {
  statements_when_false;
  ...
}
```





# Nested if-else



## ▶ Nested if-else

- ▶ If-else or if statement can be used as a subpart of another if-else or if statement.
- ▶ The else clause matches the most recent if clause in the same block.
- ▶ Nested statement can be very tricky to code.
- ▶ Indentation can improve readability

```
if (i > j)
  if (i > k)
    Console.WriteLine("A");
else
  Console.WriteLine("B");
```

```
if (i > j)
  if (i > k)
    Console.WriteLine("A");
  else
    Console.WriteLine("B");
```

Incorrect indentation

- ▶ To force the else clause to match the first if clause, you must add a pair of braces:

```
if (i > j) {
  if (i > k)
    Console.WriteLine("A");
} else
  Console.WriteLine("B");
```



# Switch



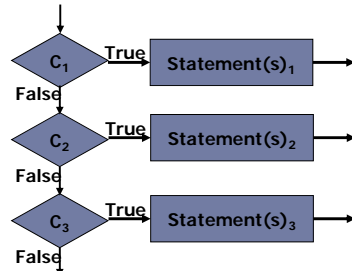
## ▶ Switch

- ▶ Acts like a multiple-way if statement
- ▶ Transfers control to one of several statements, depending on the value of an expression
- ▶ No two case statements can have the same value.
- ▶ Value must be an integer or a string
- ▶ Execution of the statement body begins at the selected statement and proceeds until the **break** statement transfers control out of the case body
- ▶ Implicit fall through from one case to another if a case statement has no code.
- ▶ The default statement is optional

```
if (Boolean_Expression)
  Statement_1;
else if (Boolean_Expression_n)
  Statement_n;
else
  Statement;
```

```
switch (n) {
  case value1:
    statement1;
    break;
  case value2:
  case value3:
    statement2;
    break;
  ...
  default:
    statementn;
    break;
}
```

Implicit fall through



# Multiway if-else



- ▶ The multiway if-else statement is simply a normal if-else statement that nests another if-else statement at every else branch
- ▶ The Boolean Expressions are evaluated in order until one that evaluates to true is found. If none of them are true then the else block is executed.
- ▶ The final else is optional

```
if (Boolean_Expression)
  Statement_1;
else if (Boolean_Expression)
  Statement_2;
else if (Boolean_Expression_n)
  Statement_n;
else
  Statement;
```

```
if (Boolean_Expression) {
  Statements_1;
  . . .
} else if (Boolean_Expression) {
  Statements_n;
  . . .
} else {
  Statements;
  . . .
}
```

## ▶ Note:

- ▶ It is indented differently from other nested statements. All of the Boolean\_Expressions are aligned with one another.



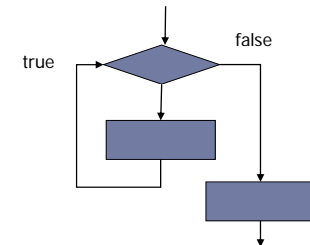
# Writing Loops

## ▶ Loops

- ▶ Repeated execution of one or more statements until a terminating condition occurs
- ▶ Pre-test and post-test loops

## ▶ Types of loops:

- ▶ Pre-test loops
  - ▶ while
  - ▶ for
  - ▶ foreach (cover in Arrays)
- ▶ Post-test loop
  - ▶ do...while



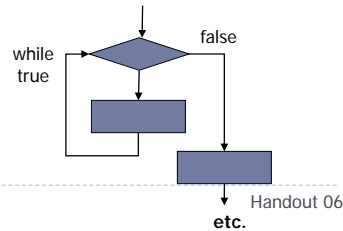


## while



- ▶ The while statement is used to repeat a portion of code (i.e., the loop body) based on the evaluation of a Boolean expression
  - ▶ The Boolean expression is checked before the loop body is executed
    - ▶ When false, the loop body is not executed at all
  - ▶ Before the execution of each following iteration of the loop body, the Boolean expression is checked again
    - ▶ If true, the loop body is executed again
    - ▶ If false, the loop statement ends
  - ▶ The loop body can consist of a single statement, or multiple statements enclosed in a pair of braces { }

```
while (boolean_expression) {
    Statement_1;
    Statement_2;
    . . .
    Statement_Last;
}
```

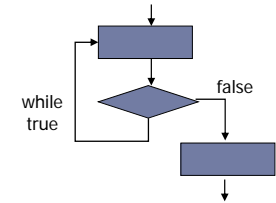


## do-while



- ▶ The do-while statement is used to execute a portion of code (i.e., the loop body), and then repeat it based on the evaluation of a Boolean expression
  - ▶ The loop body is executed at least once
    - ▶ The Boolean expression is checked after the loop body is executed
  - ▶ The Boolean expression is checked after each iteration of the loop body
    - ▶ If true, the loop body is executed again
    - ▶ If false, the loop statement ends

```
do {
    Statement_1;
    Statement_2;
    . . .
    Statement_Last;
} while (boolean_expression);
```



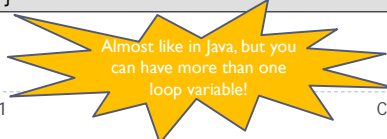
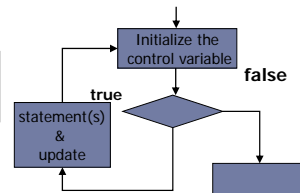
- ▶ Note: Don't forget to put a semicolon after the Boolean expression



## for

- ▶ The for statement is most commonly used to step through an integer variable in equal increments
  - ▶ It begins with the keyword for, followed by three expressions in parentheses that describe what to do with one or more controlling variables
    - ▶ The first expression tells how the control variable or variables are initialized or declared and initialized before the first iteration
    - ▶ The second expression determines when the loop should end, based on the evaluation of a Boolean expression before each iteration
    - ▶ The third expression tells how the control variable or variables are updated after each iteration of the loop body
    - ▶ The body may consist of a single statement or a list of statements enclosed in a pair of braces { }

```
for (initialization; boolean_expression; update) {
    Statements_Body;
}
```



## Note:



- ▶ Counter
  - ▶ Variables called counters are frequently used to control loops
  - ▶ Counters are invariably initialized before the loop begins
  - ▶ They are also usually modified within the body of the loop
  - ▶ The counter in the body of the loop must eventually make the test expression false
    - ▶ Otherwise, the loop will continuously loop forever - called an infinite loop
  - ▶ If you declare the control variable in the for loop, the scope of that variable will only be inside the loop block
- ▶ When to use...
  - ▶ while
    - ▶ Use the while loop when you wish the loop to repeat as long as the test expression is true
  - ▶ for
    - ▶ The for loop is primarily used when the number of required iterations is known
  - ▶ The post-test loop (do-while) is ideal when you want the loop to always iterate at least once

```
//case 1
int i = 1;
while (i < 3) {
    i = i + 1;
}
```

```
//case 2
int i = 20;
while (i < 3) {
    i = i + 1;
}
```

```
//case 3
int i = 20;
do {
    i = i + 1;
} while (i < 3);
```

```
//case 5
int i = 1;
while (i < 3) {
}
```

Infinite loop!

```
//case 4
for (int i = 1; i < 3; i++);
```



## Nested Loops

- ▶ Loop can be nested
  - ▶ When nested, the inner loop iterates from beginning to the end for each single iteration of the outer loop
  - ▶ There is no limit in how many levels you can nest loops. It is usually not more than three levels.

### Examples

- ▶ Multiplication table
  - ▶ row = 3, col = 4

1	2	3	4
2	4	6	8
3	6	9	12

i	j	output
1	1	1
	2	1 2
	3	1 2 3
	4	1 2 3 4
	5	-
2	1	1 2 3 4
		2
	...	
3	4	1 2 3 4
		2 4 6 8
		3 6 9 12
	5	-
4	-	-

```
for( int i = 1; i <= row; i ++ ) {
  for (int j = 1; j <= col; j++) {
    Console.Write( i * j < 10 ? " " + i * j : " " + i * j );
  }
  Console.WriteLine(); // Print blank line
}
```

```
for( int i = 0; i < row; i ++ ) {
  for (int j = 0; j < row; j++) {
    Console.WriteLine("");
  }
  Console.WriteLine();
}
```

```
***
***
***
```

```
for( int i = 0; i < row; i ++ ) {
  for (int j = 0; j <= i; j++) {
    Console.WriteLine("");
  }
  Console.WriteLine();
}
```

```
*
**
***
```



## break & continue

The break statement terminates the closest enclosing loop or switch statement in which it appears.

- ▶ Control is passed to the statement that follows the terminated statement, if any.

```
for( int i = 0; i < row; i ++ ) {
  for (int j = 0; j < row; j++) {
    if ( i+j >= row )
      break;
    Console.WriteLine("");
  }
  Console.WriteLine();
}
```

```
***
**
*
```

The continue statement passes control to the next iteration of the enclosing iteration statement in which it appears.

- ▶ It must be enclosed by a while, do, for, or foreach statement
- ▶ It applies only to the innermost statement in nested iteration statements

```
for( int i = 1; i <= row; i++ ) {
  if ( i % 2 == 0 )
    continue; // Go back to for
  Console.WriteLine("i = " + i );
}
```

```
i = 1
i = 3
```

i	for	output
1	1<=4	i=1
2	2<=4	Continue, jump to next iteration
3	3<=4	i=1 i=3
4	4<=4	Continue, jump to next iteration
5	False	



## goto

▶ The goto statement transfers the program control directly to a labeled statement.

- ▶ Used in a switch statement
- ▶ Get out of deeply nested loops

```
Case 1
Case 2
switch (x) {
  case 1:
    Console.WriteLine("Case 1");
    goto case 2;
  case 2:
    Console.WriteLine("Case 2");
    break;
}
```

```
int[,] array = { { 1, 2, 3 }, { 4, 5, 6 } };
for (int i = 0; i < x; i++)
  for (int j = 0; j < y; j++)
    if (array[i, j]==x)
      goto Found;
Console.WriteLine("The number {0} was not found.", x);
goto Finish;
Found:
  Console.WriteLine("The number {0} is found.", x);
Finish:
  Console.WriteLine("End of search.");
```



## return & using

### return

- ▶ The return statement terminates execution of the method in which it appears and returns control to the calling method.
  - ▶ It can also return an optional value.
  - ▶ If the method is a void type, the return statement can be omitted.

```
public static void returnMethod(int row) {
  for (int i = 1; i <= row; i++) {
    if (i % 2 == 0)
      return;
    Console.WriteLine("i = " + i);
  }
}
```

i	for	output
1	1<=4	Print i=1
2	2<=4	Return

### using

- ▶ Defines a scope, outside of which an object or objects will be disposed of.
  - ▶ It is usually best to release limited resources such as file handles and network connections as quickly as possible.

```
using (Font font1 = new Font("Arial", 10.0f)) {
}
```



## using directive

- ▶ The using directive has two uses:
  - ▶ You can reference types in the library without fully qualifying the type name
  - ▶ To create an alias for a namespace.

```
using System;
namespace B {
    public class Program1 {
        static void Main(string[] args){
            Console.WriteLine("Hello");
        }
    }
}
```

using directive

```
namespace B {
    public class Program2 {
        static void Main(string[] args){
            System.Console.WriteLine("Hello");
        }
    }
}
```

```
using C = System.Console;
namespace B {
    public class Program3 {
        static void Main(string[] args){
            C.WriteLine("Hello");
        }
    }
}
```

using alias



## Create Your OWN library

- ▶ Using Visual Studio .NET Command prompt
  - ▶ You can use /target:library option to cause the compiler to create a dynamic-link library (DLL) rather than an executable file (EXE).
  - ▶ You must use /reference:xxx.dll option to reference the assemblies that contain the Class1 class
    - ▶ Example: Class1.cs

```
namespace A {
    public class Class1 {
        public int x;
    }
}
```

csc /target:library Class1.cs

- ▶ To compile the library source code:
- ▶ Example: Program.cs

```
using System;
using A;
namespace B {
    public class Program {
        static void Main(string[] args){
            Class1 c = new Class1();
            c.x = 10;
            ...
        }
    }
}
```

- ▶ To compile the program with reference to the library:

csc /reference:Class1.dll Program.cs



## Cont'd

- ▶ Using Visual Studio .NET 2008/2010
  - ▶ To generate the DLL file
    - ▶ Open the project's Properties page.
    - ▶ Click the Application property page.
    - ▶ Modify the Output type property to DLL
  - ▶ To create reference to a DLL file in your application
    - ▶ Select Add Reference from the Project menu
    - ▶ Click the Browse Tab page
    - ▶ Select the DLL file and click OK.

```
public class Program {
    static void Main(string[] args){
        Class1 c = new Class1();
        c.x = 10;
        ...
    }
}
```

