VT Academy
Thinking Ahead, Beyond Boundaries

Object Oriented Programming

Basic Programming in C#

Session 2

Objectives

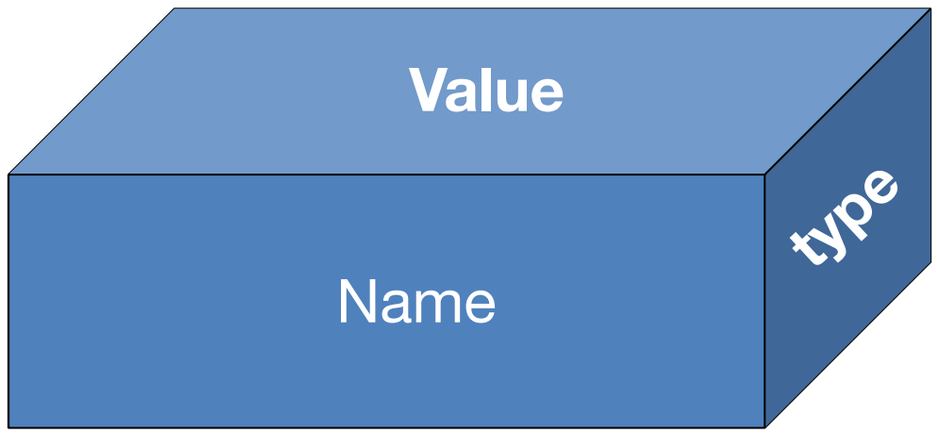
- Variables, data types and operators
- Console Input / Output
- Condition statements
- Loop statements

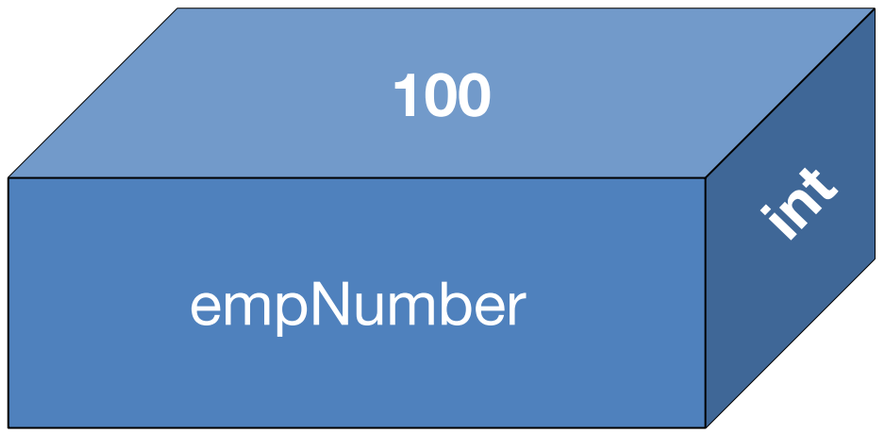
Variables

- A variable is nothing but a name given to a storage area that our programs can manipulate
- Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable

Using Variables

- In C#, memory is allocated to a variable at the time of its creation
- You can initialize the variable at the time of creating the variable or a later time
- In C# variables enable you to keep track of data





Defining Variables

- Syntax for variable definition in C# is:

```
<data_type> <variable_list>;
```

- Example:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Initializing Variables

- Variables are initialized (assigned a value) with an equal sign followed by a constant expression:

```
variable_name = value;
```

or variables can be initialized in their declaration:

```
<data_type> <variable_name> = value;
```

- Example:

```
int d = 3, f = 5;     /* initializing d and f. */
byte z = 22;          /* initializes z. */
double pi = 3.14159; /* declares an approximation of pi. */
char x = 'x';         /* the variable x has the value 'x'. */
```

Types of Variables

- The variables in C#, are categorized into the following types:
  - Value types
  - Reference types
  - Pointer types

Value Types

- Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**.
- The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating point numbers, respectively.
- When you declare an **int** type, the system allocates memory to store the value.

Value Types

VTC Academy
Thinking Ahead, Beyond Boundaries

Type	Represents	Range	Default Value
bool	Boolean value	True or False	FALSE
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

Reference Types

- The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.
- In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.
- Example of **built-in** reference types are: **object**, **dynamic,** and **string**.

Implicit Type

- Beginning in Visual C# 3.0, variables that are declared at method scope can have an implicit “type" - **var**.
- An implicitly typed local variable is strongly typed just as if you had declared the type yourself, but the compiler determines the type.
- The following two declarations of i are functionally equivalent:

```
var i = 10; // implicitly typed
int i = 10; //explicitly typed
```

Reference Types

- The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.
- In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.
- Example of **built-in** reference types are: **object**, **dynamic,** and **string**.

```
object obj;
obj = 100; // this is boxing
string str = "Hello";
```

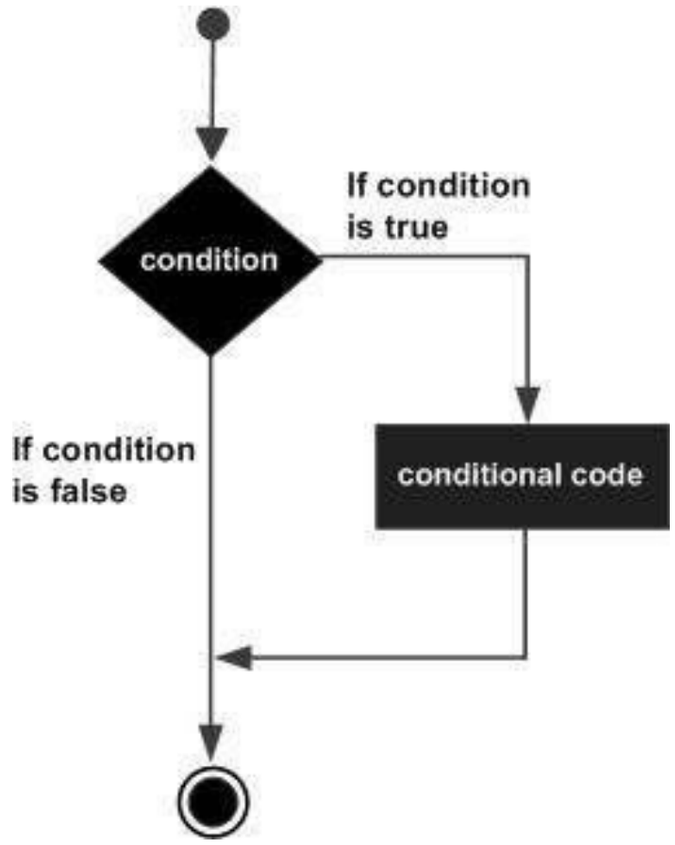
Condition Statements

- Condition statements require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

- C# provides following types of condition statements:
  - if statement
  - if..else statement
  - switch statement

if Statement

- An **if** statement consists of a boolean expression followed by one or more statements. If the boolean expression evaluates to **true**, then the block of code inside the if statement is executed.



- Syntax:

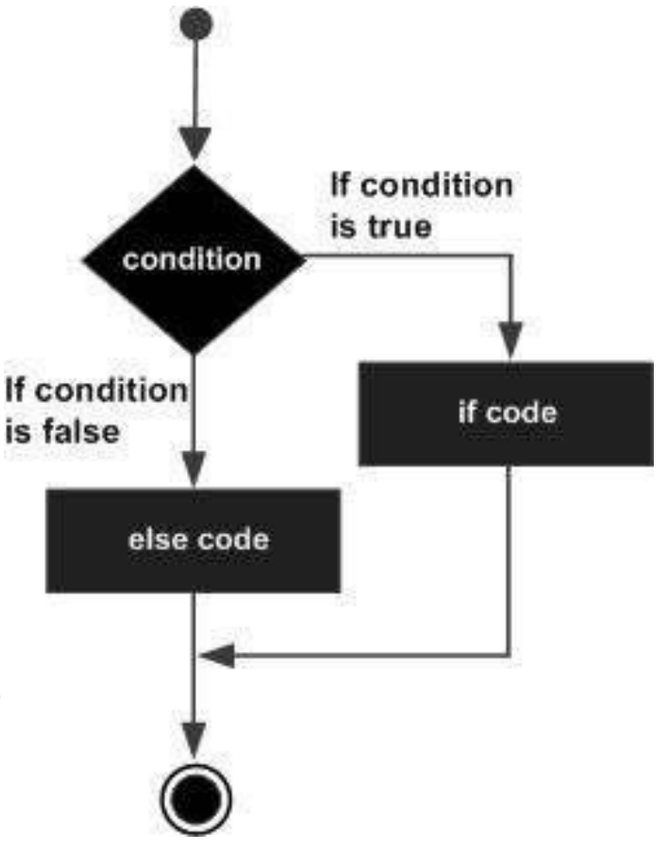
```
if(boolean_expression)
{
   /* code will execute if the boolean expression is true */
}
```

if Statement

```
using System;
namespace IfDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         // local variable definition
         int a = 10;
         // check the boolean condition using if statement
         if (a < 20)
         {
            // if condition is true then print the following
            Console.WriteLine("a is less than 20");
         }
         Console.WriteLine("Value of a: {0}", a);
         Console.ReadLine();
      }
   }
}
```

if..else Statements

- An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false. If the boolean expression evaluates to **true**, then the **if block** of code is executed, otherwise **else block** of code is executed.



- Syntax:

```
if(boolean_expression)
{
   // code will execute if the boolean expression is true
}
else
{
   // code will execute if the boolean expression is false
}
```

if..else Statement

VTC Academy
Thinking Ahead, Beyond Boundaries

```
using System;
namespace IfElseDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         // local variable definition
         int a = 100;
         // check the boolean condition
         if (a < 20)
         {
            // if condition is true then print the following
            Console.WriteLine("a is less than 20");
         }
         else
         {
            // if condition is false then print the following
            Console.WriteLine("a is not less than 20");
         }
         Console.WriteLine("Value of a: {0}", a);
         Console.ReadLine();
      }
   }
}
```

switch Statement

- A **switch** statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each **switch case**.
- Syntax:

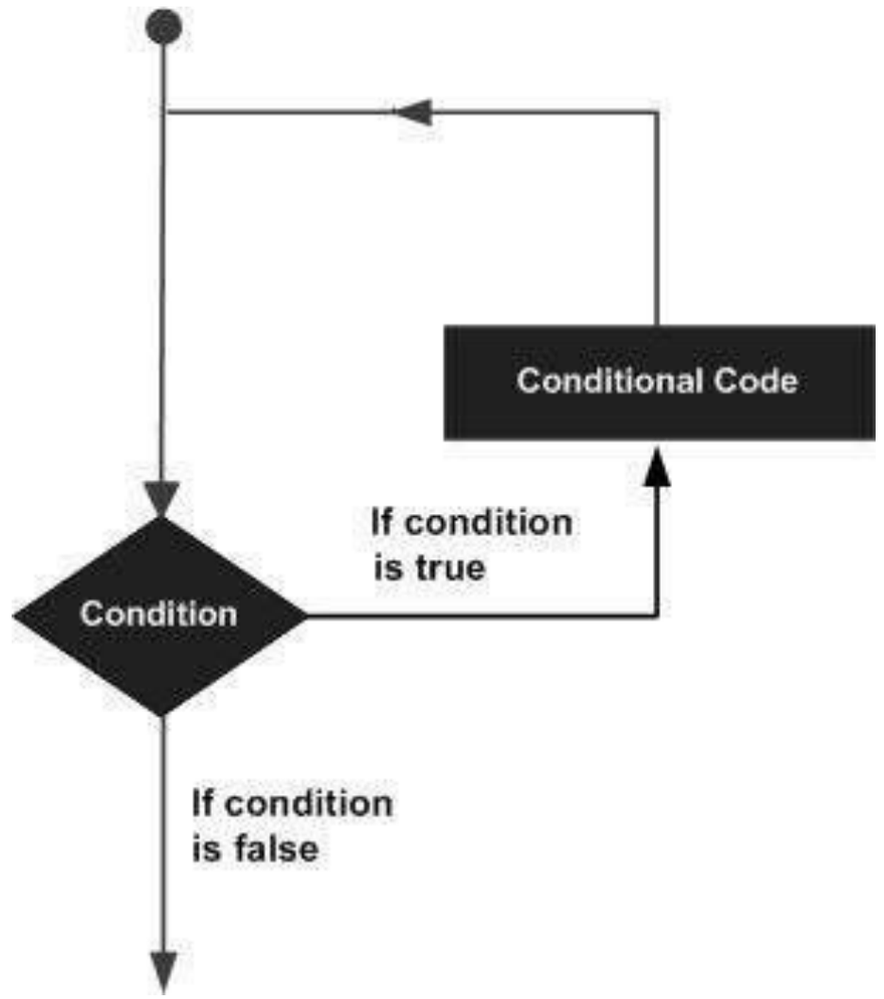
```
switch(expression)
{
   case constant-expression:
      statement(s);
      break; // optional
   case constant-expression:
      statement(s);
      break; // optional
   // you can have any number of case statements
   default: // optional
   statement(s);
}
```

switch Statement

```
using System;
namespace SwitchDemo {
   class Program {
      static void Main(string[] args) {
         char grade = 'B';
         switch (grade)
         {
            case 'A':
               Console.WriteLine("Excellent!");
               break;
            case 'B':
            case 'C':
               Console.WriteLine("Well done!”);
               break;
            case 'D':
               Console.WriteLine("You passed!”);
               break;
            case 'F':
               Console.WriteLine("Better try again!”);
               break;
               default:
            Console.WriteLine("Invalid grade!”);
               break;
         }
         Console.WriteLine("Your grade is {0}", grade);
         Console.ReadLine();
      }
   }
}
```

Loop Statements

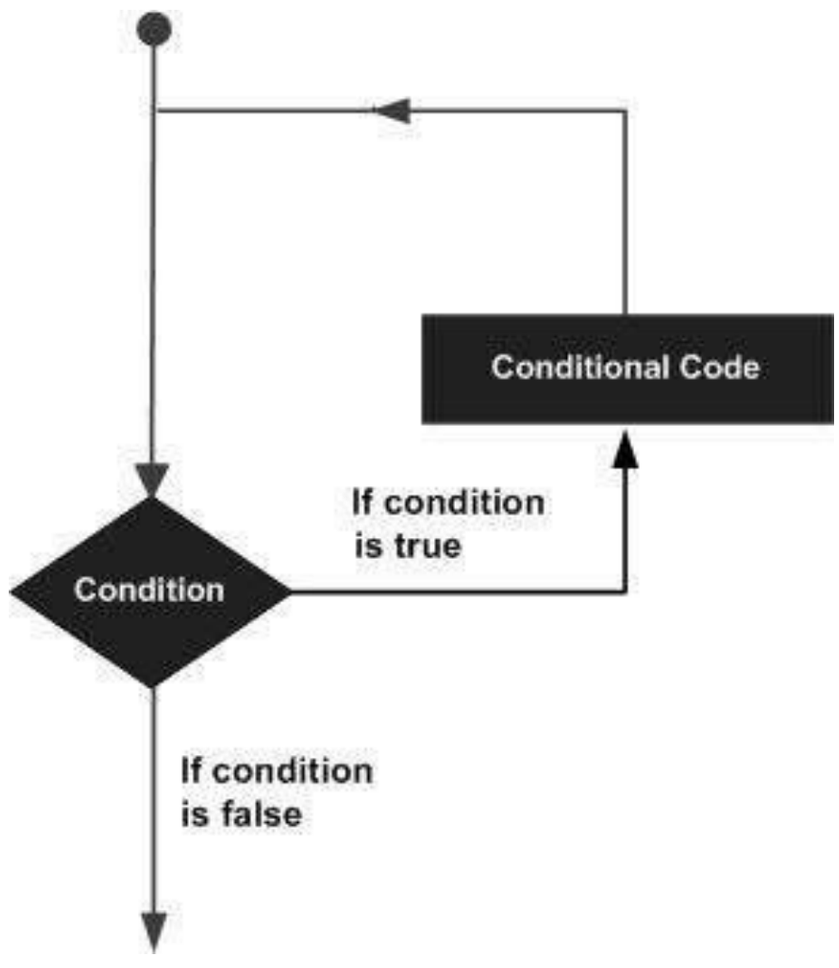
- A loop statement allows us to execute a statement or a group of statements multiple times until a condition still true



- Types of loop statements:
  - **for** statement
  - **while** statement
  - **do**..**while** statement

Loop Statements

- A loop statement allows us to execute a statement or a group of statements multiple times until a condition still true
- C# provides following types of loop statements:
  - **for** statement
  - **while** statement
  - **do**..**while** statement

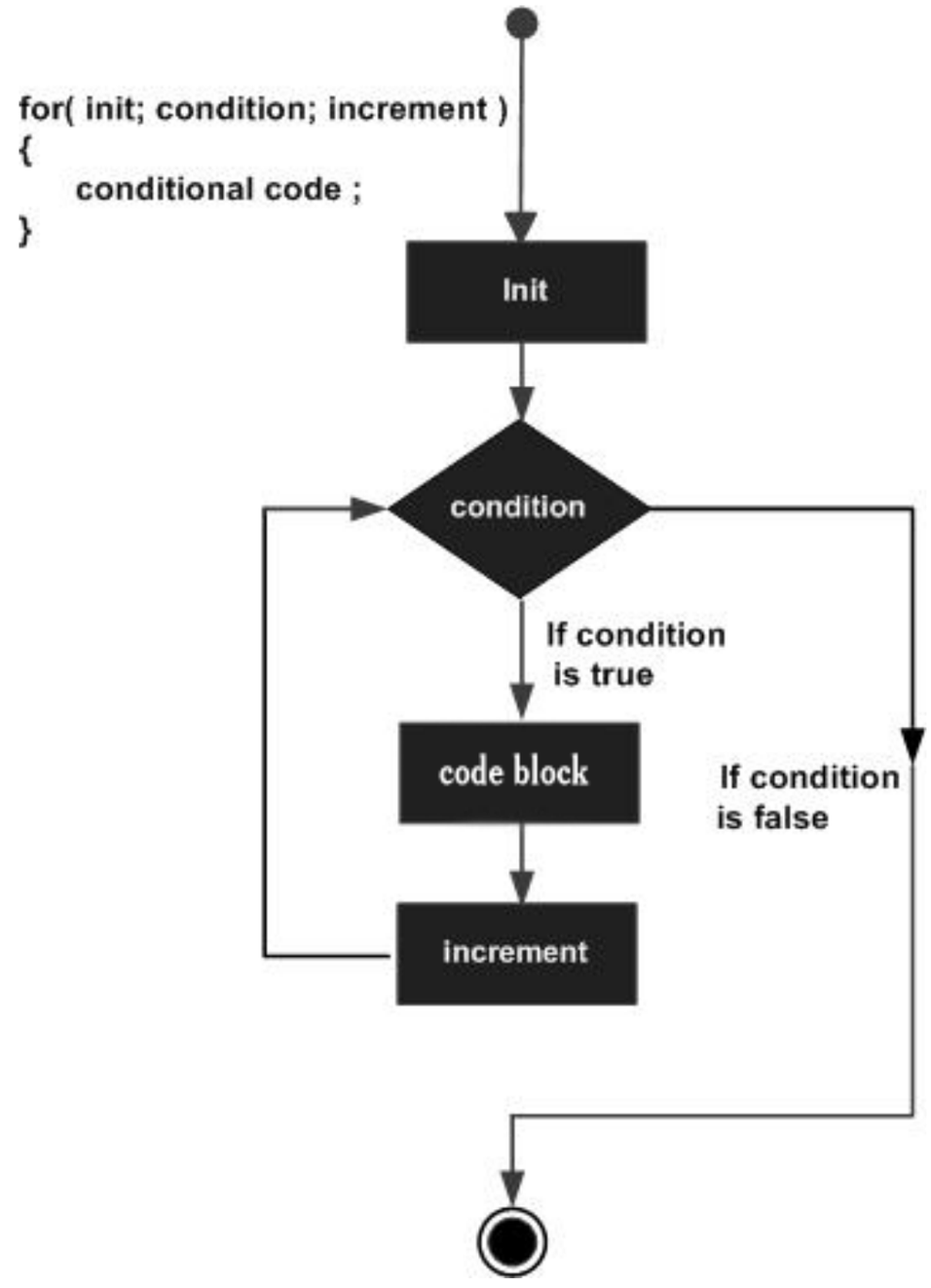


for Statement

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times

- Syntax:

```
for (init; condition; increment)
{
   statement(s);
}
```

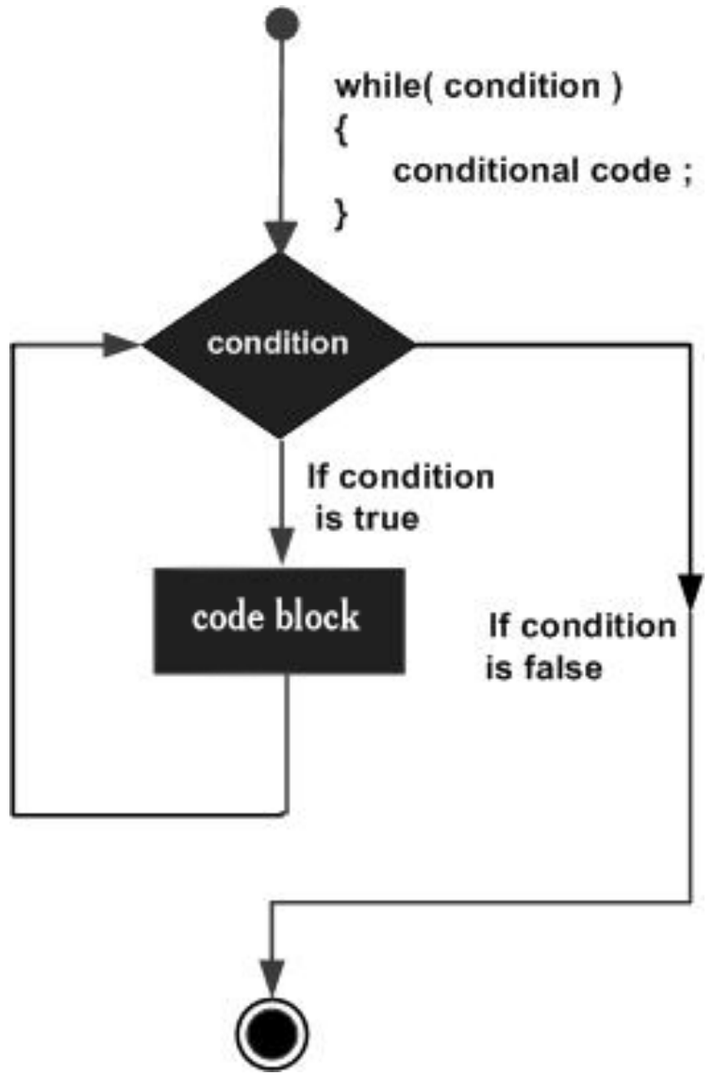


for Statement

```
using System;
namespace ForDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // for loop execution
            for (int a = 10; a < 20; a = a + 1)
            {
                Console.WriteLine("Value of a: {0}", a);
            }
            Console.ReadLine();
        }
    }
}
```

while Statement

- A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true



- Syntax:

```
while(condition)
{
   statement(s);
}
```

while Statement

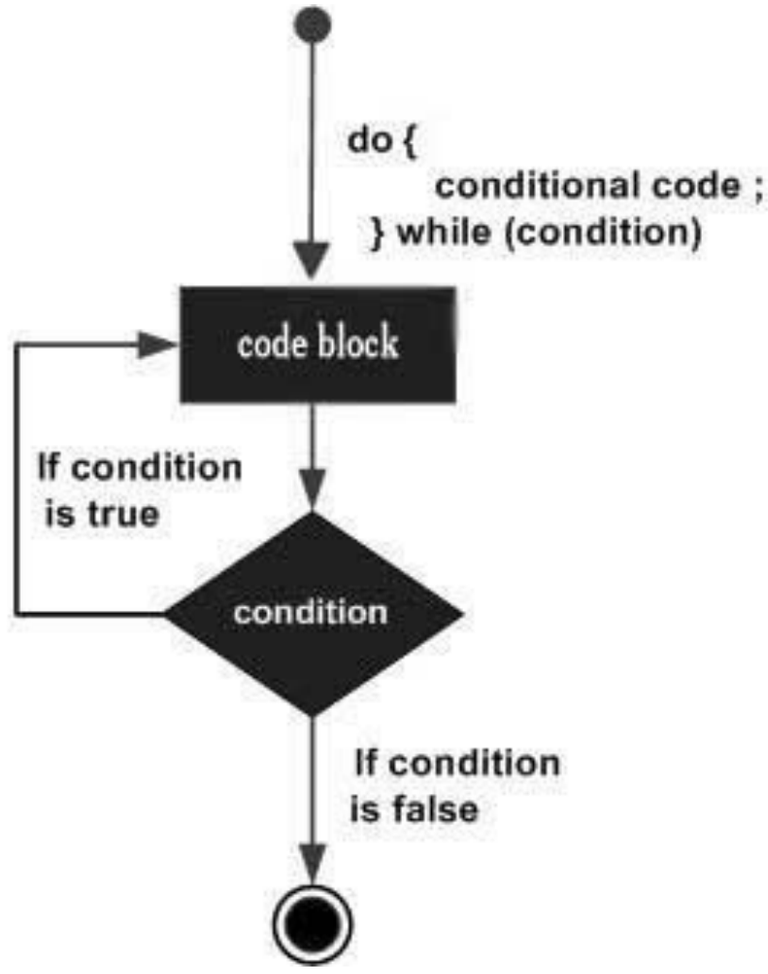
VTC Academy
Thinking Ahead, Beyond Boundaries

```
using System;
namespace WhileDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // local variable definition
            int a = 10;

            // while loop execution
            while (a < 20)
            {
                Console.WriteLine("Value of a: {0}", a);
                a++;
            }
            Console.ReadLine();
        }
    }
}
```

do..while Statement

- Unlike **while** loops, which test the loop condition at the start of the loop, the **do...while** loop checks its condition at the end of the loop
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time



- Syntax:

```
do
{
    statement(s);

}
while(condition);
```

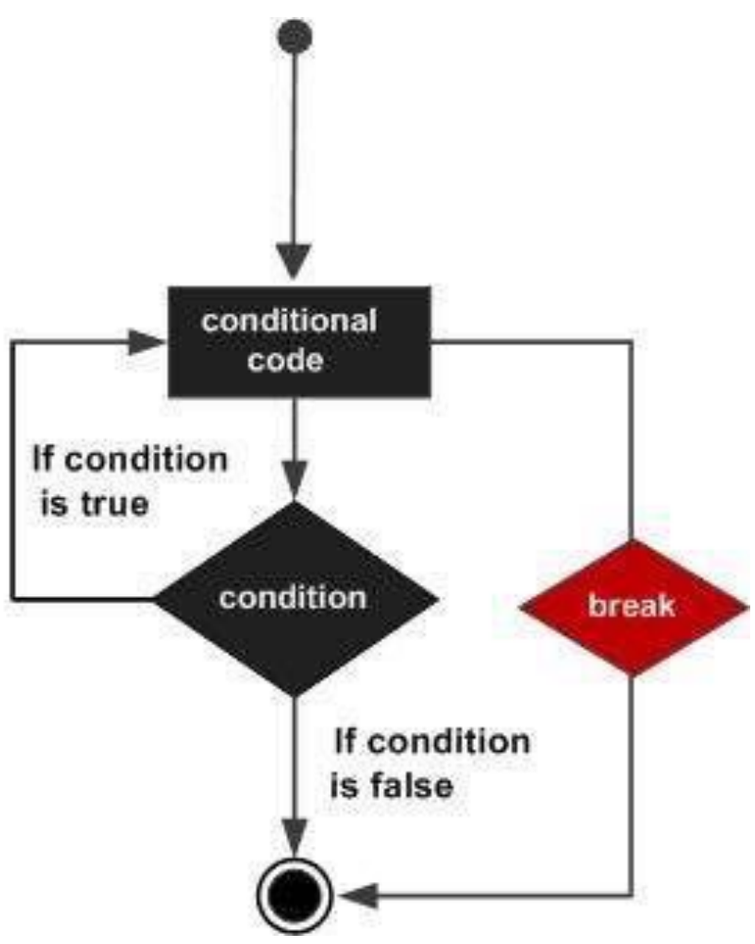
do..while Statement

```
using System;
namespace DoWhileDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // local variable definition
            int a = 10;

            // do loop execution
            do
            {
                Console.WriteLine("Value of a: {0}", a);
                a = a + 1;
            }
            while (a < 20);
            Console.ReadLine();
        }
    }
}
```

break Statement

- The **break** statement in C# has following two usage:
  - When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop
  - It can be used to terminate a case in the **switch** statement

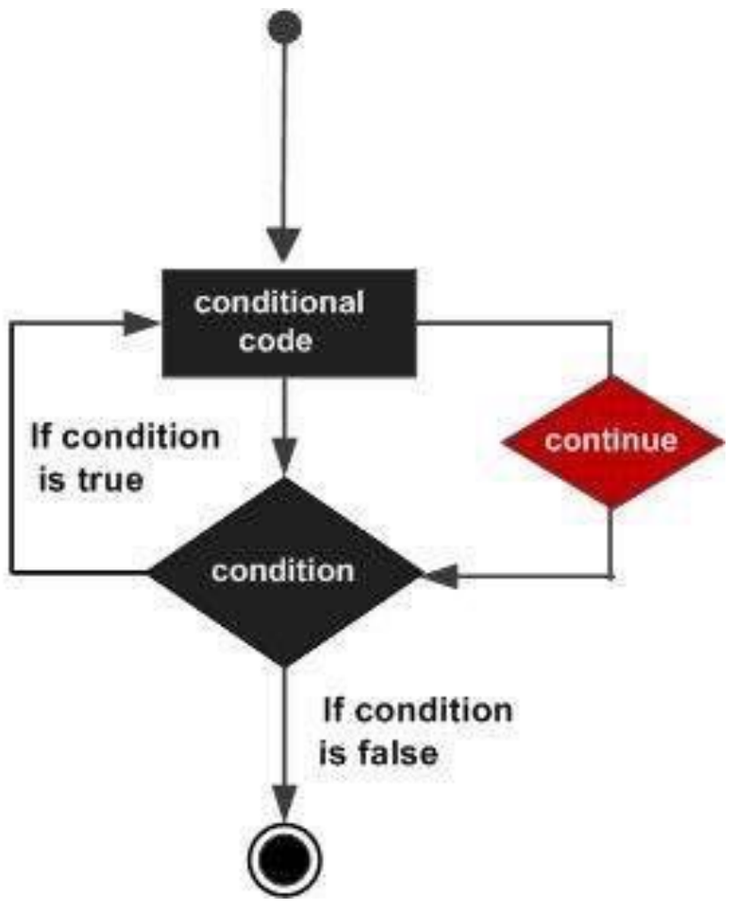


break Statement

```
using System;
namespace BreakDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            // while loop execution
            while (a < 20)
            {
                Console.WriteLine("value of a: {0}", a);
                a++;
                if (a > 15)
                {
                    // terminate the loop using break statement
                    break;
                }
            }
            Console.ReadLine();
        }
    }
}
```

continue Statement

- The **continue** statement in C# works somewhat like the **break** statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.



continue Statement

```
using System;
namespace ContinueDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            // do loop execution
            do
            {
                if (a == 15)
                {
                    // skip the iteration
                    a = a + 1;
                    continue;
                }
                Console.WriteLine("Value of a: {0}", a);
                a++;
            }
            while (a < 20);
            Console.ReadLine();
        }
    }
}
```

Summary

- Each variable in C# has a specific type and size
- The variables in C#, are categorized into the following types: value types, reference types and pointer types
- Beginning in Visual C# 3.0, variables that are declared at method scope can have an implicit “type" - **var**
- C# provides following types of condition statements: if, if..else, switch statement
- C# provides following types of loop statements: for, while, do..while statement