C#

STEP BY STEP
BEGINNER'S
GUIDE

PROGRAMMING BASICS FOR ABSOLUTE BEGINNERS

NATHAN CLARK

C#

Programming Basics for Absolute Beginners

Nathan Clark

Table of Contents

This document is presented with the desire to provide reliable, quality information about the topic in question and the facts discussed within. This eBook is sold under the assumption that neither the author nor the publisher should be asked to provide the services discussed within. If any discussion, professional or legal, is otherwise required a proper professional should be consulted.

This Declaration was held acceptable and equally approved by the Committee of Publishers and Associations as well as the American Bar Association.

Any information contained in the following pages is considered accurate and truthful and that any liability through inattention or by any use or misuse of the topics discussed within falls solely on the reader. There are no cases in which the Publisher of this work can be held responsible or be asked to provide reparations for any loss of monetary gain or other damages which may be caused by following the presented information in any way shape or form.

The following information is presented purely for informative purposes and is therefore considered universal. The information presented within is done so without a contract or any other type of assurance as to its quality or validity.

Any trademarks which are used are done so without consent and any use of the same does not imply consent or permission was gained from the owner. Any trademarks or brands found within are purely used for clarification purposes and no owners are in anyway affiliated with this work.

Introduction

Congratulations on purchasing *C#: Programming Basics for Absolute Beginners* and thank you for doing so.

The following chapters will discuss how to get started with the C# programming language and how you are going to be able to use this for your own personal programming needs. Many people are worried about being able to use a programming language in their own lives. It may look complicated or they feel that all the rules and moving things around will just make it too hard to handle. But C# is a great option because it is simple to use, works with Windows devices, as well as several others, and you won't look at the rules and feel confused right from the start.

This guidebook is going to take the time to look at some of the different aspects of C# programming and even go through creating a few different codes so that you can get some practice. Sometimes getting into the coding process and giving it a try while learning, can make things so much easier to understand and avoids all the confusion and worry in the process. C# is a very easy program to use and you are going to love how easy it is to get started with.

In addition to learning how this code works, you will learn all the other basics that come with using C# programming language. We will talk about variables, constants, comments, and so much more. Whether you just want to get a bit of experience with the programming world or you are ready to increase your computer programming knowledge and learn something new, C# is a great option to get started with and you are going to love how easy it is to use. This guidebook will give you all the tips that you need to understand this language and to get started right away.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

The Basics of C#

When it comes to getting started with programming languages, many people run away in fear. Programming is hard and difficult for some people to understand and unless you have some experience or have worked with computers before, it is difficult to always understand what is going on in the instructions. This chapter will start with some of the basics that you need to get down in order to really learn about C# and how to use this great programming language.

Why Use the C# Language?

There are a number of programming languages that you can use to make your own codes from home. All of them have benefits and negatives that will determine whether you want to use them or not. Sometimes it is a matter of how clean the code looks and other times it may be based on the type of computer you are using or the program that you want to design.

In this guidebook, we are going to spend some time looking at C# and showing all the great things that you can do with this programming language. Even as a beginner, there are things that you can learn with this program that will make the process easy while still providing some of the bells and whistles.

So why would you choose to go with the C# program compared to going with some of the other choices such as Java, C++ and Python? Here are some of the benefits that you should consider:

- Have lots of functions in the library—if you are just getting started and want to learn how to write out functions, there are a ton of them inside of the C# library so you will be able to use whenever you need some help. You can place these inside the code if you need it, or even make changes to have it work in your particular code.
- Automatic disposal of functions—most programming languages will require you to go through and manually delete any item that you don't want there. This can take a lot of time and will waste space on your memory if you don't take care of it enough. But with C#, you will find that the program will delete the functions and objects that don't have a value attached to them. This will help you to clear out the memory on your computer and saves a lot of time and space on your computer.
- Easy to work with—C# is one of the easiest programs for you to work with. While it may not be as simple of a language as Python, it is on the easier side and you will find that the code is not clunky and you will be able to read through it pretty easily once the code is done. There are also options for writing comments so you will be able to leave some clarification in the code if someone else needs to use it later.
- Does well with Windows computers as well as others—this programming language was originally used in order to work with Windows computers and help you to design a program for them. But it

also works well with some other systems such as Mac, Linux, and more as long as you download .NET on it. Windows has some great products that are easy to use, especially for beginners, so you are sure to get great results when you get started.

- Works with .NET which helps make it easy—this is a program that already comes with the Windows computers, but you can add it to some of the other systems in order to make C# accessible on these computers as well.
- Similar to C and C++. This makes it really easy to work with this program and learn the basics before going on to these other programs. Even if you choose to stick with this programming language, you are going to find that it is powerful enough to do most of the coding that you want without all the hassle of some of the other programs.

While you have a lot of choices to make when it comes to picking out a coding language that will help you to make some great programs while still being simple enough to learn as a beginner, the C# is one of the best ones that you can choose from for all the great benefits and things that you can do with it.

What You Need

Let's start out with some of the things that you will need in order to use this programming language. First, the computer you are using should have a ".NET" framework as well as the development kit that is used for Windows computers. Most of the modern operating systems from Windows, with those starting on Windows Vista and above, will have this framework already on the computer so you won't have to install anything new on your computer.

If you are working on a computer that doesn't have the .NET framework, you are able to download it for free. You just need to go visit www.microsoft.com and then search for the framework to get started.

One thing to note is that if you plan to use a Linux, UNIX, or Mac computer, you will need to choose Mono Project rather than the .NET framework. This kit will help you to create programs with C# on these computers. You can go to www.monoproject.com to get this proper framework.

Learning the C# Language

When it comes to using a Windows computer, C# is one of the best programming languages that you can find. It is compatible with the .NET framework that is on most modern Windows computers and this language is simple, flexible, and still has a lot of power with it. If you are new to programming and like the simplicity of using Windows products, then the C# language is the right one for you.

Just like working with the Java language, C# isn't going to support any code pointers or even multiple inheritances. Rather, it is going to offer type checking and memory collection. It also has some of the powerful features that programmers like from C++, such as overloading, enumerations, and preprocessor directives so you can get the simplicity and the power that you are looking for out of this program.

Trying Out Your First Program with C#

The best way to learn some about programming with C# is to try writing some of the programs. So let's dive in and get started writing a simple program with this language to help you get your feet wet and learn a bit more about this option:

- Launch the text editor of your choice. If you are on a Windows computer, Notepad is a great option. If you are on Linux, vim or vi are good options to try out.
- Type the following code:

```
class FirstProgram
{
    static void Main()
    {
        Console.WriteLine("Using C# is fun.");
    }
}
```

- Now you can access the command prompt and type:

```
csc FirstProgram.cs
```

After you issue this command, the compiler for C# will process this file and then create a .exe file in the same location as your code. For example, if you saved the original file on the desktop, you should see a new program come up called "FirstProgram.ee." right in the same place. If there is an error in the code that you wrote, you will see an error message come up.

Now you can run this application by entering "FirstProgram.exe." into the command prompt. If everything was done correctly, you should see that the

command prompt will display the message: “Using C# is fun.”

Analyzing This Program

As you can see, creating a program on this language is not too complicated. You will need to go through a few steps, but you can make the message and the results as simple or as complicated as you want. Now that you have had a chance to great your C# program, it is time to go over what you just did (if you notice there were quite a few lines there for just writing out the simple statement) so that you can understand this program later.

- The first line—this is the line that will consist of the keyword and the identifier. The keyword is going to be a word that will have special functions inside the language. With the keyword above, you are creating a class for the program. The identifier is going to help you to identify the method, variable, and class that you want in the program. "FirstProgram" is the name of the class you are creating in this situation.
- In the third line, you are defining the method. Main() will act as the starting point for any application on the computer. Programs will start executing on the Main() method, no matter where they are placed inside the code. There are usually two words that will go before the method, mainly void and static. In this case, static is going to inform the language compiler that the method will not create an object within the class. When using void, you are telling the compiler that Main() won't return a value.
- In the fifth line, you will see the final part of the code. This is the line where you will write your message. The method known as WriteLine() captured the combination of characters (or the sentence) and will then print it on the screen when you request.

In addition, you probably also noticed that there were braces in place that were used to tell the computer that there are code blocks throughout the code to help separate things out. Keep in mind that C# will use a semicolon (as noted in the code above) just like Java and C.

C# Comments

This programming language will allow you to leave some comments within the code. This is going to be a line that will provide some more information regarding the code that you added. The compiler is going to ignore the lines so your comments will have no bearing on what comes out on the computer. But this can be a great way to remember what you were trying to do in the code or to show other programmers what you are working on.

C# will allow for two types of comments. The first one is the single line comment. This kind will start with two forward slashes. The amount of words that you say should be pretty small so you can keep them on a single line. If you go over to more than one line, you will notice that the program sends you an error message. A good way to write this out is:

```
static void Main()
{//I have a pet.}
```

You can also use multi-line comments. Sometimes you can't keep the comment to just one line and you will need to add in something different to allow the code to go over two or more lines. Luckily, C# does allow you to write these kinds of comments, as long as you begin it with the symbols /* and end it with */. An example of this is:

```
static void Main()
{
/*The C# compiler should ignore this line.*/
}
```

This makes it easier to write a longer piece of comment, in order to explain things in more detail, without having it come up on the screen in your code or having an error sign come up when you are working on the code.

Comments can be your best friend when it comes to working on the code. It is going to help others know what they are supposed to do with the code, such as letting them know what they should put within the brackets at a particular part, explain what you are working on with that code, and more. You should be careful with the comments and not necessarily put them after each line of the code, but using them when needed can make things easier.

Things to Remember With C#

Some of the things that you should remember when it comes to creating a C# program include:

- All C# programs will exist inside of a class.
- You should start all of the executions that you want inside the program using Main().
- This programming language will be case sensitive. This means that class and CLASS will have different meanings to the program.
- The compiler is going to ignore the white spaces that you create. This includes spaces, tabs, and new lines. You can use this whitespace as much as you like to help make the program look a little easier to read.
- C# will allow you to choose any name that you want for the programs. You don't need to have the program's name and the class identifier be the same.
- You can use Main() more than once within the program.
- When defining the boundaries of a method or a class, make sure to use the curly brackets.
- You can use two types of comments within C# to help explain things better. Multi-line and single line are fine, as long as you use the right features of each.
- You can always add in an argument or a string to your Main() method.

Using C# is one of the best programming languages that you can choose. It is great to use with most Windows computer, and it works great with other computers as well. It has all the great features that you have come to know and love with Java and C++, but it is much easier to use and can work for beginners without all the experience!

Variables and Data Types

With a first glance at using C#, you can see that this language is not too scary. While some of the other options are meant for more advanced users and can be a bit complicated, you will find that this is not always the case when using C#. So now that you have gotten your feet a bit wet with C#, it is time to delve in deeper and learn a few more things to make it more fun.

Data Types

Those who have used C# often will choose to divide the programming language into two parts; reference types and value types. When you are working with a value type, you are going to pass the data over to whichever method you are using. On the other hand, when working with a reference type, you should just add the reference in with the method as the value will be placed somewhere else. Some of the data types that you will find when using the C# program include:

- Ulong
- Double
- Long
- Float
- Uint
- Int
- Ushort
- Sbyte
- Short
- Byte
- Bool—when you use this type, the values that you are allowed to store can only be two. The values are going to be false and true so it is good for conditional statements and logical expressions.
- Decimal
- Char—this is a data type that is only allowed to have one single character. When you are writing the char value, you must include it with single quotes such as writing out ‘t’ ‘b’ and so on.

Variables

When you are doing an execution, the computer program is going to store all the data for a short amount of time. Programmers will often use the word "variables" when they are referring to the locations of memory that will hold this stored data. Thus, the variable will have a data type and corresponding value. When you are working in the C# language, you are able to create a variable using this syntax:

<data_type> <name_of_variable>

The line above will save a part of your computers' memory in order to hold onto the char value. The programmer will then be able to access this variable using the identifier it was given. For example, if you had listed the above formula as char x, you could find the variable by going to x.

C# also allows you to initialize the variables during declaration. This term refers to the process of giving a value to your variable when you are creating it. You just need to use your assignment operator, which is =, and then indicate the value that you are trying to assign here. Some examples of this could be:

int year=2015

bool isItDelicious = false

char sample = '4"

You can also declare a few variables for one statement as long as they are of the same data type, you just need to separate each of the entries using a comma. An example of this is:

char ant = 'a', bull='b', cat='c'

Just like with other programming languages, C# will require you to declare

your variables before you are able to assess them again. C# is also going to implement a new rule called definite assignment. This means that you will need to initialize the local variable before you are able to use it. You basically need to assign the initial value of a local variable during the declaration process.

Something to keep in mind: variables are named this way because the values that they are going to store will change every time that the program runs. The values are going to change based on what you place into them.

Constants

Next on the list to understand are constants. These are variable types that will prevent the program from changing the initial value. This means that the value is going to stay constant, no matter what the user is inputting in the process. You will need to use "const" when you want to declare this constant. An example of this is:

```
const char LETTER=x;
```

Before you state one of these constants, you need to make sure that you really want it that way. The program will not be able to change the value of a constant if you've already put it in place. This means that you must assign the initial value to it right at the variable declaration. For example, you will get an error message if you put in the following command:

```
const bool ANSWER;
```

One thing to keep in mind about constants is that most are going to use uppercase letters when they are declaring this. Knowing this information can help you to read through C# code a bit easier and will make it easier to create your own in a manner that others will be able to read and understand.

Creating Your Identifier

If you look through the suggestions from Microsoft, you will find that they recommend that you use a Camel notation when you are working with variables and a Pascal notation when working with methods. When you are doing a Camel notation, you will see that the first letter of the word is lower case. If you are dealing with a compound word, the first letter in your second word will start with an uppercase letter. Some examples of this include:

payment completePayment

mathematics firstClass

The Pascal notation will take things a bit differently. This notation style is going to ask you to start the first word using an uppercase letter. The first letter in all of the other words in the sequence should also be done in uppercase as well. Some examples of using the Pascal notation include:

WriteLine() ReadLine()

Start() Main()

It is also possible to use underscore characters and numbers as well when you are creating an identifier. But, you are not allowed to start the identifier using a number. You can write fiveBooks as an identifier, but you are not allowed to write 5Books.

While these notations aren't mandatory, you are often advised to use them when you are writing statements with C#. This can help you to keep the code pretty clean and makes it easier for others to be able to go through and read your source codes later on if they need to.

C# has made creating a code pretty easy. You don't need to worry about a lot of complicated rules that come with some of the other codes and yet you are still going to get a lot of the power that you are looking for. The good news is, we have already looked at some of the most important parts that come with

using C#, and so far you have even written a code. So now let's take this knowledge and move into some more of the things that you are able to do with this great programming language.

Working on the Operators

The next thing that we will learn about are the operators. The operators are going to help your created programs learn how to perform tasks and handle values. Without this part of the formula, your computer programs can't do much and will become useless so knowing how to write operators and how they work will make a big difference in how your program works for you.

Just like with other programming languages that you may work with, C# has operators that will belong in different categories. Let's take a look at each of the categories and how you will handle the operators in each one.

Arithmetic Operators

These are the operators that will tell your computer program to do an arithmetic procedure. This can help the computer know when you need to add things together, subtract them, and so much more. The computer can do some of the basics for you while making the right signs to show the program what you would like it to do. Some of the arithmetic operators that you can use in C# are:

For these examples let's assume that x=15 and y=10

- "+" this is the addition operator. It is going to add two operands together. So you would get x+y=25
- "-" this is the subtraction operator. It is going to allow you to subtract the value of the right-hand operand from the left-hand operand. So you would get y-x=-5.
- "*" this is the one that will tell the computer to multiply the two operands. So you could do x*y=150.
- "/" this one is the operator that will tell the computer to divide the left-hand operand with the right-hand operand. For example y/x.
- "%" this is often called the remainder or the modulo operator. It is going to divide the left-hand operand by the right operand and then returns the remainder.
- "++" this is the increment operator. It is going to increase the value of the operand by one. So you would have ++x=16
- "—" programmers will often call this the decrement operator. It is going to decrease the value of the operand by one. This means that you will have –x=14.

One thing to note is that the decrement and the increment operators are considered unary, which means that it can only be used on one operand. Also, you can also add the operators together before or after using the increment and decrement values.

Here is an example o a code that has arithmetic operators inside:

```
class Example
{
    // This basic program will show you how arithmetic operators work
    static void Main()
    {
    Int f =3, g = 4;          // This line declares two variables, f and g.
    sampleSum = f+g;          // This will give you 7.
    sampleDifference = g-f          // This will give you 1.
    sampleProduct = f*g;          // This will give you 1.3
    sampleModulo = g%f;          // This will give you 1
    f++               //This will give you 3.
    }
}
```

Using arithmetic operators is pretty simple as it is just going to tell the computer to put two numbers together. You have to set up the parameters for what all the numbers will equal and the computer can do the rest.

Adding in the comments can help someone else who is looking at the program figure out what you are doing, but you don't have to add these in if they are pretty basic numbers. Sometimes the numbers get a bit more complex so it is good to get in the habit of using these comments even on the easy ones so that other programmers, and even yourself, will be able to read through them.

Assignment Operators

Next on the list is assignment operators. This is an operator that will allow you to assign a value to your variable. Some of the assignment operators that you can choose from when using C# include:

- "=" this operator will allow you to perform simple assignment operations. It is going to assign the value to a variable that you are working on at that time. For example, writing *int sample = 100* will tell the program that you want to assign 100 to the variable that is called "sample". It won't perform any extra processes on this variable or on the value involved.
- "+=" this is the additive assignment operator. It is going to add up the values of your two operands and then will assign the sum to your left-hand operand.
- "-=" programmers will often refer to this as the subtractive assignment operator. It is going to subtract the value of the operand on the right side from the one on the left side and then assign the difference to the left-hand operand.
- "*=" this operator will multiply the values of each operand and then will assign the product to the left-hand operand.
- "/=" this is when you will divide up the two variables and then take the result and assign it to the variable on the left.

One thing to note is that when you use an assignment operator, you will need to make sure that both operands do belong to the same type of data. If you find that the operands are incompatible, the program may not work properly during runtime and you will have to go back through and make the changes.

Relational Operators

These are the operators that will let you compare the values of your two operands. Because of this, they are good when used for conditional statements. Some of the relational operators that you may encounter when using the C# language.

For these, let's assume that d = 100 and e = 150

- "==" this is the operator that you can use to check the equality of two values. If the two values end up being equal, the operand will tell you it is true. Otherwise, the operand will tell you it is false. For example, saying the d == e would show up as false.
- "!=" this operator allows you to test the inequality of two values. If the values end up not being equal, it will tell you this is true. For example e != d would result in a true.
- ">" this operator is used to check whether the operand on the left is greater than the operand on the right. If it is, then the operator will tell you it is true. For example, saying that e > d would be true.
- "<" this is the less than operator, it will allow you to check whether the operand on the left side is less than the operand on the right side. If it is, you will get it to show up true, such as the formula d < e.
- ">=" this is the operand that will say it is true if the value of the operand on the left side is greater than, or equal to, the operand on the right side. Otherwise it will tell you the statement is false. For example, saying that e >=d evaluates as true.
- "<=" with this operator, you will get a true if the operand on the left side is less than or equal to the operand that is on the right side. For example d <= e is true.

When you are using the relational operator, the result is to get a Boolean value each time. This means that you want to get an answer that is either true or false each time. You should also check to see if you are using two equal signs any time that you are using the equality operator. If you get this mixed up with the

assignment operator, you are likely to get an error in the program or it just won't work right for you.

Logical Operators

Another option that you are able to use with operators is logical operators. This is sometimes called a Boolean operator because it is going to accept two Boolean values in order to produce a brand new Boolean value. Keep in mind that C# will allow and support four types of logical operators.

For this we are going to assume that c = true, d = true, and e = false

- "&&" this operator is called logical AND. It will only result in a true if both operands are true. For example d && c will evaluate to true.
- "||" this is the logical OR. This operator is going to give you a true if at least one of your operands is true. For example, c || e will result in true.
- "^" this operator is the Logical Exclusive OR and it will result in a true if one of the operands is true. If both operands can be false or true, the operator will give you a false.
- "!" with this one, you will be able to reverse the value of your Boolean variable. For example, if you type in !d, you will get a false.

Some programmers will refer to || and && as short circuit operators. This is because these signs are able to give accurate results without checking the whole expression. An example of this is &&, since it will require two trues, it will result in a false if the first operand says false, no matter what the rest say and it won't even look at them. This can help to speed up the program a bit and get the work done.

Bitwise Operators

This kind of operator is going to work pretty similarly to the logical operator. The difference is that the bitwise operator will take binary values in order to produce the Boolean result. Since this operator is going to work on binary options, basically values that are composed of zeros and ones, they are going to show their results with either a 0 or a 1 as the output. The C# language is able to support the following bitwise operators.

For this, let's assume that I = 0, h = 1, j =0 and g = 1

- "&" this is the operator that is known as the Bitwise AND. It is going to assign 1 to the positions where both of the operands have 1. For example g & h will give you 1.
- "|" this is the bitwise OR. It is going to assign 1 to the positions when there is at least one of the operands with a 1. For instance, doing h | 1 will give you a result of 1.
- "^" this is the exclusive OR that will work well for binary data. Just like working with the logical values, this kind of operator is only going to give you 1 in the areas that the operand has a 1. For example, if you do g^j you will get a result of 1.

These operands may seem a little tricky to deal with in the beginning, but they are going to make it so much easier to do some of the different operations that you need when working in C#. They are simple to use and after a little bit of practice, you will find that they are easier than ever to use and not quite so complicated. Perhaps consider opening up your text reader and seeing how well these work for you, experimenting a bit to get the results!

Conditional Statements Used

Conditional statements are another part of the C# language that can help you and other programmers to create flexible and powerful applications. In fact, this kind of testing has been considered a very important part of writing any kind of computer program so that you can get the code right for others, as well as yourself, to use correctly. If you want to be really effective and good at using C# language, you need to have some mastery with conditional statements.

If

The first conditional statement that you can use is the if statement. This is one of the most basic conditional statements that you can use in C# and it is going to allow the computer to behave according to certain inputs that you place in. the syntax of using the if statement includes:

If (The Boolean expression)

{

The statement/s you want to run

}

As you can see, this is one of the simpler formulas that you can use and it will be changed based on the length of the statement that you are trying to run as well as the other conditions that you want to have in this expression. Some of the main parts of the if statement includes:

- The "if" keyword—this is the part that will tell your language compiler that you are putting together an "if" statement.
- The Boolean expression—this is the part of the syntax that will determine if the program will run the body or not. You are not able to use the char and int values for this kind of expression when using C#.
- The statement you want to run—this is the body of the syntax and you are allowed to have one or more statements put inside. The program will be able to execute these statements if the Boolean expression results in a true. If the result ends up being false, the program is going to ignore the statements and will pass this control over to the other statements that you have written out.

Don't let this seem too confusing for you. Here is a good example of what all of this means:

```
If (x > 0)
    {
    Console.Write("The value is positive.");
    }
```

With this example, the program is only going to print "The value is positive' only if your value is higher than zero. If you find that the expression is false, then the program will ignore the Console.Write part of the statement and will move on to the next part of the program.

If Else

Building upon this is the if else clause and can allow you to do a lot of great stuff with this program. A syntax that you should remember for the "if else" statement includes:

If (the Boolean expression)

{The statement/s you want to run if the result is true;

}

Else

{The statement/s you want to run if the result is false;

}

There are five parts that come with this kind of expression and it can really enhance what you are able to do with the program, including the ability to have two different classes of statements show up depending on what the answer is. Some of the parts include:

- The "if" keyword—with this keyword, you are telling the compiler that you are going to make a conditional statement.
- The Boolean expression—the expression that you are making should result in a Boolean value. If the expression ends up being true, the program is going to run the first statement that you listed.
- The first body—this is the statement that you want to have run if the Boolean expression ends up being true. You can add in a few statements here, just make sure they all fit within the same bracket.
- The "else" keyword—this is the keyword that will tell the compiler that there is an "else" clause. This is the statement that is going to run if the previous Boolean expression ends up being false.
- The second body—this one is pretty similar to the first body. But if the Boolean is false, this is the one that will run rather than the first one. You can choose to have just one statement or several statements

depending on your needs.

As you can already see, the “if else” statement is going to come out more powerful than the if statement. With the if statement, if the answer is false, you aren’t going to have anything show up and the next command in the language is going to be displayed. But with the “if else” you can pick which statement comes up regardless of whether you get a true or false answer, making the code much more powerful.

Let’s take a look at how the “if else” statement could work in real life.

```
If ( x > 0)
    {
    Console.Write(“This value will be positive.”);
    }
    Else
        {
        Console.Write(“The value is less than or equal to zero.”)
        }
```

In this syntax, the “else” clause is going to remain hidden unless the Boolean expression ends up being false. It is there if the process needs it or the value comes out false, but if the value is true, the first statement is going to be used and the second one will be ignored completely.

The “if else” clause is one that a lot of beginners won’t use. They think that it is too complicated and won’t help them to get things done. They want to stick with just the “if” clause and assume that it is the best option for getting things done. But for those who want to have a lot of options with their statements, or who want different options to show up depending on the answers that are given, using the “if else” clause is going to be one of the best options that you

can use.

It isn't that hard of one to understand. It just provides you some more options than you will find with the other choices. After you get some time to experiment with the "if" clauses, make sure that you work on getting used to the "if else" clauses so that you can bring your new codes to a whole new level.

Nested Conditional Statements

When you are using the C# language, you will be able to write an "if" statement inside of another "if" statement when it is necessary. This is a process called nesting and it is used to help you to create complex programs that rely on chained conditions. One of the things that you should keep in mind when using this option is that you will run into syntax errors and compile time if you make a mistake and do the writing part of the nested conditional statements the wrong way.

Many experts in C# recommend that you should limit your nested statements to just three levels. If you go above this amount, you could end up with a code that is confusing and complex, making it hard for others to read and resulting in more errors in the process.

A good example of a code that allows nesting for a conditional statement includes:

```
Double r = 60;
Double s = 70;
If (r ==s)
{
    System.Console.WriteLine("These numbers are equal.);
    }
    Else
    {
        If (r > s)
          System.Console.WriteLine("The value of the first variable is greater than that of the second one.")'
        }
        Else
```

```
    {
        System.Console.WriteLine(The value of the second variable is greater than that of the first one.")'
    }
}
```

With this option, you have a few different scenarios that come up using the "if else" clause. This allows you to have a few different options just up for the command so that the program knows what to read out. This can help you to do so much more with the program than before, but be careful not to make it too complex or you could end up with a mess.

Switch-Case

These switch-case statements will execute a code according to the expressions' results. Most programmers with C# will use integer values when working on this kind of statement. To create one of these statements, it is recommended to use the following syntax:

```
Switch (value_selector)
{
    Case integer1;
        The statements you want to execute;
        Break;
    Case integer2;
        The statements you want to execute;
        Break;
    //...
    Default:
        The statements you want to execute;
        Break'
}
```

There are a few parts that you will need to understand in order to really get the most out of this syntax. These include:

- The "switch" keyword—this is the keyword that will inform the compiler that you want the program to perform a switch-case statement.
- The value selector—this is the expression that will generate a value. With the C#, the value should be compatible with your operators. Your program is going to compare this value against the other cases that are listed inside your statements.

- The “case”—the case is going to be the label that used by your program when it does comparisons. If the case ends up matching the value from the selector, the statement will run. The program will check each case until it is fund.
- The executable statements—these are the different statements that you will want to run based on the case that is appropriate. It can be a group of statements or a single statement based on your needs.
- Break keyword—this is the keyword that will terminate the body of a switch structure. You will need to place this keyword after each statement that you are trying to create.
- The default clause—this is the mechanism that you need in order to end the statement. It is going to run if the selector doesn’t match any of the other labels that you put up there. This pretty much ensures that the process shows up something if nothing else works out. You can make sure that the program will put out an answer no matter what the inputs are.

Working with conditional statements can sometimes be difficult for a beginner. You will find that there are so many things that you are able to do with these kinds of statements that it can seem a bit intimidating for a lot of people. But when you see some of the syntax options and experiment with using these options on occasion, you will find that it is easier than you can think to do some great options with language programming.

The important thing to remember during this is to use the different sections that are listed above. This will help you to ensure that you get all the right parts in order to work on the conditional statement. Without the right options, you will find that the program will bring out errors signals when you try to run it. When you get through with all of these new aspects, you can write a great conditional statement that will help you get your program up and running.

Cr eating and Using Objects

With this chapter, we are going to focus on two important programming concepts, mainly classes and objects. You are going to learn how to access all the classes that are present in the .NET programming framework as well and it has some really valuable information that can make it easier than ever to help you become a master of C# without all the hassle.

Classes and Objects

First, we are going to talk a bit about objects in this programming language. Programming has had a lot of growth over the past few years and all this growth has had a huge effect on how programmers are going to create their new computer applications. Object Oriented Programming, or OOP, is one of the newest and biggest ideas that has been developed in the IT world and we are going to spend some time learning how OOP works so that you can use it to your advantage to create some amazing programs with C#.

Object Oriented Programming

This kind of programming is a style that will rely on objects. OOP will provide you with a model that is based on how things will work in the real world. When you are able to work with objects rather than with abstract ideas, it is easier to understand the language and get it to work for you. This approach is going to help you to solve the problems that come up in programming using your intuition and logic.

Objects

Programmers are going to use digital "objects" in order to represent physical objects or abstract ideas. While using the OOP, you should remember that these objects will have two characteristics including:

- State—this is the characteristics that will define the object. These can be general or specific.
- Behavior—this is the characteristics that will declare all the actions that an object is able to do.

Let's take a look at the difference between these two characteristics by looking at an actual object—a ball. The state of the ball is going to be the size, color, and make of the ball. The behavior of the ball could be something like rolling or bouncing.

With the OOP, you will be able to combine the information and the techniques to process them into one thing. The programming object is going to correspond

to the actual one and will hold onto the information and the actions.

Classes

When using C# the classes will define the characteristics of your object. They will give you a structure or a model for defining the nature of the object. According to some programming experts, the classes are basically the foundation of OOP and they will be linked closely with the objects. In addition, each of the objects that you use will represent one particular class.

Let's start with a class that is called "Toys" and the object that is "Ball." In this instance, the "Ball" is just one instance of the "Toy" class. But the "Toy" class will define the behavior and the state of all the toys as well as of the ball.

Classes can add some simplicity to your computer programming. The information they will hold onto should be meaningful for anyone who looks at the program, not just professional programmers. For example, the class of "Toys" can't have HTTP as one of the characteristics since these can't be linked together.

Attributes and Behavior

Your classes are going to define the behavior of an object, such as the actions that the object is able to perform, and the attributes, or the characteristics. The attributes are going to appear as different variables in the body of the class while the behavior is going to be defined by the different methods inside the class.

Let's apply these to the Toy class. When they are applied, you will get the color and the size as attributes. For this you are going to use the methods stop() and move().

Using Classes in a Program

At this point we have discussed a bit about classes and how they work, but we need to get a bit more into how they are going to work within the program and how they can make a difference in what you are trying to write out in the code. With C#, you should define all of the classes using the keyword "class" to keep things simple.

After you type in "class". You will need to indicate the particular identifier that you would like to use along with the variables and the methods that you would like to place inside this new class. To keep things simple, the different parts that you can add into the class in C# language include:

- Fields—these are any of the variables that will belong to a particular data type.
- Methods—you can use these methods in order to manipulate the data
- Properties—in this language, the properties are going to enhance how well the fields work. The property will provide some extra management abilities for the information and give it to the fields.

Below is an example of the different parts that we just discussed so that you can get a good idea of how this would work in a programming language. We are going to use "book" in the name of this class and give it two properties: size and type.

```
Public class Book
{
    Private string bookType:
    Private string size;
    Public string BookType
    {
    Get
    {
```

```
Return this. bookType;
}
Set
{
    This.bookType = value'
}
}
    Public string Size
    {
        Get
        {
        Return this size;
        }
        Set
        {
        this.size = value;
        }
        }
        public Book()
        {
        this.bookType = "Dictionary";
        this.size = "large";
        }
        public book(string bookType, string size)
        {
        this.bookType = bookType;
```

```
        this.size = size;
        }
        public void Sample()
        {
        Console.WriteLine(" Is this a {0}, bookType)
        }
}
```

With this example, the Book class is able to define two different properties, the book type and the size. These properties are able to hide the values inside the fields of the same names and the code snippet is going to declare two constructors for generating on the Book class. This code also created a method called Sample().

You will be able to use this for a number of different things, adding in more characteristics if you would like, and it doesn't have to be limited to books, toys, vehicles, or anything else. You can mess around with your text editor a bit to try out this formula and change it around with some other classes to make it easier to get the program to work for you.

System Classes

The C# language already has a library that is built into the system. These are going to include some default classes like Console, Math, and String. As you use this language, you must keep in mind that this library is going to be compatible with any of the .NET applications that you use.

The .NET framework is nice because it already has this preinstalled library that will include various classes inside. The classes can be helpful for those who are trying to do some basics in programming tasks, like text procession, execution, and networking. They can be helpful for the beginner who is just getting started or even someone who has been working in the programming language for some time because they are simple and already written out.

Keep in mind that these classes are going to hide the logical implementation inside. You should focus more on what the classes are able to do, not exactly on how they are able to do it. For this reason, the built-in classes with C# won't be viewable to the programmer. Since you should worry more about the principle of working on your code rather than all the abstract parts, it is important to just use these codes for their general purposes rather than worrying about how they work.

Creating and Using Objects

We have spent some time looking at classes so now it is time to work on creating and then using objects in the language. Here are some of the basics that can help you to get started:

Creating an object

To get started with creating an object inside an existing class, you will need to create a new keyword. Usually, the programmer will assign the new object a variable that will be of the same data type as the class of the object. Keep in mind that doing this won't copy the object to the variable, but it will just give the variable a reference to the object that got assigned to it. Use the following code to view how this would all work:

Book someBook = new Book();

This example is going to assign the instance of a Book class to the variable called "someBook." This will get the object to go to the right class that is needed.

Setting the parameters for your new object

C# will allow you to assign some parameters for your newly created objects. Let's take the syntax that you did above and make the adjustments for this:

Book someBook = new Book("Biography", "large")';

This code is going to be good for creating a new object named someBook and will assign the two different parameters to it as well. With this adjustment, the object's type is not Biography while the size is now large.

Whenever you are using the new keyword, the framework of .NET is going to complete two things:

- It will reserve some of the memory for this new object
- It will initialize the object's data members

This process is going to occur thanks to a method called constructor. For the code above, the initial parameters are the parameters of the class constructor. This will help the object to stay in the right place and give it the characteristics that it needs.

Releasing the object

While some of the other programming languages that you can use will require you to manually destroy objects, this is not a requirement with C#. This means that you will be able to release any of the memory that is consumed by your various objects without having to do manual deletions. The CLR system that comes with your .NET framework will make all of this possible.

With the CLR system, the computer will automatically detect and then release any objects that don't already have a reference. The memory that was assigned to these objects is not available and you can use it for other objects. Overall, this makes the system easier to use and can prevent issues and bugs. If you would like to release an object, you should take the time to destroy the corresponding reference. A good example of this is:

someBook = null;

This process won't delete the object, it is just going to remove the reference from the object so that the CLR can then go through and perform an automated deletion.

Accessing the object's field

When you want to access the field of an object with C#, you will need to use the dot operator or "." You will simply need to indicate the name of the object, place the dot, and then enter the field you are looking to access. However, you won't even need to add in the dot if the object you want to work with is just in

one class. You have to get into the object's field if you want to assign a new value or extract the value.

For those who are working with a property, you will need to use the keywords get and set. The set keyword is going to allow you to assign the value of an object while the get keyword is going to help you to extract the value out of the object.

Here is an example of a code that uses this information to get an object's property. We are going to create an object called myBook and then assign the bookType as "bible" to make it easier.

```
Class BookManipulation
{
    Static void Main()
    {
        Book myBook = new Book();
        myBook.bookType = "Bible";
        System.Console.WriteLine("This is a {0}." myBook.bookType);
            }
    }
```

Calling an object's method

You will need to use two operators including the invocation and the dot operators. You will not need to use the dot operator if the object and the method you choose to work with are in the same class, but the parentheses are always going to be mandatory. To call out the method, you just need to indicate the identifier and then follow it with the invocation operator. You can place the parameters inside the parentheses or you can add in the arguments for the method.

While you are writing your program in C#, you will be able to add in an access modifier to the methods. There are four access modifiers that are supported in C# including private, internal, protected, and public. These are

going to help you to restrict the calling abilities when you are working with your method. While all of them are important, we are just going to focus on public modifiers and how they will make the information, namely the objects, publicly available.

The Constructors

Constructors are methods that will run any time that the programmer creates a new object. The idea behind the constructor is to initialize the data of your new object. You aren't going to get anything value when using this method. Plus, the constructor is going to use the same name as the class it belongs to, meaning that it won't have a random name for you to remember. You will also be able to assign parameters to your constructors when using the C# language.

Parameters with your constructors

The constructors in C# will accept parameters just like the methods that you would use in the language. You can set up several different constructors in the classes, but you should check to make sure that the constructors have different types and numbers of parameters; this basically makes sure that the constructor is unique.

Remember that your constructor is going to run any time that you create your objects inside the class. If you have a class that has several constructors inside of it, you may be curious to figure out which constructor is running when you create a new object. C# will be able to determine the right constructor for you so you won't need to worry about this and you won't need to do anything manually.

The language compiler is going to select the right constructor based on the parameters that you set up in the beginning in a principle that is called "best match." This helps to keep things organized and can speed up the program writing process.

A good example of using constructors include:

```
public class Device
{
    private string type;
    private string size;
```

```
    // A constructor without any parameter
    public Device()
    {
        this.type =”laptop”;
        this.type = “large”;
    }
    //A constructor with two parameters.
    {
        this.type = type;
        this.size = size;
    }
}
```

These parameters can help to tell the program what you want to do. It also makes it easier for C# to figure out which constructor you want to show up based on the work that you are doing at that moment.

Static Data Members

So far we have been working on indicating the state of objects that are being used. These fields are going to be directly linked to objects that are inside a specific class. In OOP, you will come across different special types of fields and methods linked by class, but not with the object itself. This is called a "static data member" because it won't be affected by the objects. It can even function inside your class without having any objects there.

When you are writing your program, you need to take the time to define the static methods using the "static" keyword. You can place this in the method's value type or the field type. You can even use these to create static constructors if you need to in C#. For this kind, you would place the "static" keyword right before the name of your constructor.

Objects and classes are a big part of how C# works and they are going to help you to really make the code that you are looking for. The topics above are just some of the great things that you can do with your classes and objects in this programming language and with a bit of experimenting, you will find that almost anything is possible with this easy to use language!

Defining Classes in C#

We spent some time working on classes and objects in the past chapter, but here we are going to work a bit more with classes and learn how you can define these classes within the C# language. Let's take a look and get started with this part!

The Basics of Classes

In a programming language, the class is going to define the data and object types that you are able to use in the program. The object is going to contain this actual information that will define the state of its container. Classes are also able to describe the behavior of the object. This is all the behaviors that the object is able to perform. When doing OOP, you should use methods in order to describe the behavior of the objects.

Components of Classes

There are a few different parts that come with each class including:

- Declaration—this is the line that will declare the identifier of the class.
- Body—just like with methods, the classes are going to have a single body. You will need to define the body right after you make the declaration. The body is the statement, or several statements, that are found between the curly brackets. An example of this is:

```
class Example
{
//This is the body of the "Example" class.
}
```

- Constructor—this is the part that will allow you to create a new object. An example of this is:

```
Public Sample()
{
//Insert what you want to say here.
}
```

- Fields—these are the variables that you will declare within your class. The fields are going to contain the values that will represent the exact state of the object they are trying to get to.
- Properties—this part will describe the different attributes of the class. Many programmers will write the class properties right inside the field of their chosen object.
- Methods—a method is basically a named block of code that is executable. In is able to complete some tasks and then will allow

objects to attain the right behavior. It can also execute the right algorithms that are present inside the codes.

The following example is going to show you the best way to create a class using the C# language and it is going to include all of the pieces that we discussed above as important to classes.

```
class ZooAnimals        // This is the class declaration
{       // This bracket signals the start of the class's body
   string animalType        //This line declares a new field
   public ZooAnimals()       // This line declares an empty constructor.
   {
   }
        public  ZooAnimals(monkey animalType)          // This line create another constructor for the class.
   {
      this.animalType = animalType;
   }
   string    ZooAnimals        //Here, you are declaring a property.
   {
      get {return animalType;}
      set { animalType = value;}
   }
   Static void Animal()       // This line declares a new method for the class.
   {
      System.Console.WriteLine("This is a {0}." animalType ??)'
}
```

Objects and Classes

Now that we know a bit more about how to create and use class objects, it is time to learn a bit more about this technique and how it is going to work when you are programming.

Custom classes

Before you start using a specific class, you will need to have an object inside it. You can do this by starting with a new keyword and the constructors that are present inside the class.

In the C# programming language, you are not going to be able to manipulate the objects directly. This means that you will have to assign your object to a variable before you are able to manipulate it. You can then access and use the object through whichever variable you choose to assign the object too. Keep in mind that in order to access the methods and the properties of the object, you will need to indicate the identifier of the object and use the dot operator.

More about objects

Each of the objects in your .NET framework will have two parts, the reference part and the actual part. The actual part is going to contain the information about your object and it is stored inside a heap that is known as the dynamic memory on the operating system of the computer. On the other hand, the reference part of the .NET framework is going to exist in the execution stack of the program, which is the part that will hold the method parameters and the local variables.

If you want to create a new variable that isn't associated with an object, you need to make sure to give it a value of "null". This keyword is going to tell the language compiler that the variable doesn't have a value at this time.

Organizing Your Classes

When you are saving in C#, there is just one rule that you will need to follow and that is all the classes need to be saved as .cs files to make it easier to find and to make the class work. Technically, you are able to save all of the classes that you are creating into one file and the compiler will still work without any errors, but in most cases it is best to save them in different files to help you keep things organized. Sometimes it is hard to find the class that you want if you save them all in just one file.

Using namespace

You are going to want to get familiar with using namespace with C#. This is a set of classes that will be related in some logical manner. It can include classes, interfaces, structures, and other kinds of information. You can combine together a few classes into a namespace regardless of where they are located in the memory.

If you would like to use a namespace on the codes that you are creating, make sure that you add a “using” directive to make this easier. Most programmers find that it is easiest to write these directives inside the first couple of lines in the .cs files to make sure they don’t forget about it. After you insert the directive, you can declare which namespace you wish to use.

Access Modifiers

C# is able to support four types of modifiers, protected, private, internal, and public. These modifiers are going to allow you to have some control over the visibility of the elements in your class. Let's take some time to discuss each of the modifiers in detail so you know which one is right for you.

- Private—this modifier is going to place a restriction on the class. If it tagged as a private element, it is not accessible by any of the other classes. The C# will use this as the default for modifiers so if you don't decide what accessibility the element has, the system will make it private.
- Public—this is the modifier that you should choose if you want your element to be accessible with the other classes. This means that you are taking away all the limitations in regards to the visibility of the object.
- Internal—if your element has this as the modifier, it is going to be accessible just to the files that are in the same project.
- Protected—this modifier is going to prevent a user from accessing the element. However, it will all the descendant classes to access and use the elements that are involved.

Each of these modifiers is going to work in a slightly different way based on what you want it to do. If you want to keep it protected and ensure that it isn't accessed by the other classes, you will want to make sure that you use a private setting. But if you would like other elements and classes access to it, you will need to make it public. And of course, there are options that always go in between to help you determine how much access other classes will have to this main modifier.

You will need to determine which one is going to be right for your code. Most of the time you will find that private is going to be the default option, but you can always make the change to ensure that your modifiers are doing what they should be within the program.

Declaring a Class

C# does have some strict rules when it comes to class declarations. You are going to get runtime errors and compile time issues if you don't do a good job with declaring your classes. To ensure that you aren't getting any errors, you should use the following syntax:

<modifier> <class> <name_of_the_class?>

You need to write in the body of the class right after the identifier to keep things easier. The body is going to be the part that will contain the executable codes so just like with other languages for programming, you should write this inside of your curly braces.

"This" Keyword

Another keyword that you should learn about is the "this" keyword. This is a tool that will help you to access the contents of a class. Some examples of using this keyword include:

this.sampleMethod(); //Use this syntax to run a method.

this.sampleField: //Use this syntax to access a particular field.

this(8, 9); // Use this syntax to trigger a constructor that contains two parameters.

The Fields

The object is going to represent things that are in the physical world. To define this object, you are going to spend your energy concentrating on the attributes, which will then be linked to purposes of your program on the computer. You will need to store the attributes into the class declaration with the help of special variables. These will be called fields and they will define the various statuses of the object you are working with.

Declaring a field

C# is going to require you to declare all of your fields right in the body of the class. The following code will show a few examples of how you can declare fields in your class:

```
class Example
{
    double salary;
    char favoriteLetter;
    string yourName:
}
```

The scope of a field

The scope of your field will begin from the part where you start writing it and will continue until the body of the class is done. This allows for a lot of chances to write out what you would like to say in the field.

Initializing your field

C# makes it easy to set the value of any fields that you create. The syntax that you will use to set the value of the fields will be pretty similar to the syntax that you use for ordinary variables. The syntax that you should use for these fields looks like this:

<modifier> <type_of_field> <name_of_field> = <value>;

Something to take note of is that you should check that the initial value is going to be compatible with the field that you are using. C# is kind of strict on the rules regarding how compatible the data types need to be in order to avoid issues with errors and runtime issues coming up on the screen. Take a look at the following example:

```
class Sample
{
    int payment = 5000;
    char letter = 'x';
    string [] instruments = new string[] {"guitar", "drums"};
    Vehicles myVehicle = new Vehicles()
    }
```

This is going to help you to put together the field that you would like in the program. You can make changes that you would like to make this function work the best with your chosen code.

Default Choices with the Field

Any time that you create your object inside a class, the system is going to set aside some memory in order to hold the fields of the new object. The system will be able to do this by setting an initial field or the value of every field. If you don't go through and set this default the way that you want it, the .NET framework will work to set the values for you.

This is the main difference that occurs between the local variables and the fields. The language compiler is going to send you an error message when the local valuables don't have the right value or any value at all. So you need to make sure to go through this and get it set up with a value early on or you may run into trouble with your project and you will have to go through and make the right changes.

Customizing your default value

When you are working with the C# language, you will be allowed to set the default values for any of your fields. This is a great feature that is going to allow you to make the code easier to read and much cleaner compared to some of the other programming languages that you will choose. Let's take a look at some of the fields that you saw earlier and see how we can make them customized so they look better within the program:

```
class Sample
{
    int payment = 0;
    char letter = null;
    string [] instruments = null;
    Vehicles myVehicle = null;
}
```

Readability and cleanliness are two of the most important parts that come with

using the C# code so you need to find ways to work with the code, value, classes, and other parts to compact them and make it look as nice as possible without having too much work going on all the time. Working with setting your default value can really help you to get this done right.

Working in programming can be intimidating to some of those who haven't had a chance to work in it, but perhaps they have visions of really complicated codes that look like a mess and are too difficult to mess with. The C# code is a great language to learn with because it is simple to use and has a lot of great features that even a beginner is going to be able to learn about with ease.

Conclusion

Thank for making it through to the end of *C#: Programming Basics for Absolute Beginners*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is to get started on using C# programming language in your own life. This is a great product to use, a programming language that works on most computers and is made to be simple to use and read for beginners. We have walked you through some of the steps that you need to take to create a program with C# and even discussed some of the most important parts of the C# programming language so you can really get started on the right track.

While many other programming languages are great to use and can allow you to do some powerful things, some of them are a bit confusing and can scare away someone who is brand new to the whole process. But with C#, you will not have this issue. You will be able to get started on your first program right away (and we discuss some of the options of doing this inside the book) and you will find that programming can be a lot of fun and really easy.

When you are ready to get started with programming and making it work for your needs, make sure to check out this guidebook and learn just how easy it is to work with C#.

Finally, if you found this book useful in any way, a review on Amazon is always appreciated!

About the Author

Nathan Clark is an expert programmer with nearly 20 years of experience in the software industry.

With a master's degree from MIT, he has worked for some of the leading software companies in the United States and built up extensive knowledge of software design and development.

Nathan and his wife, Sarah, started their own development firm in 2009 to be able to take on more challenging and creative projects. Today they assist high-caliber clients from all over the world.

Nathan enjoys sharing his programming knowledge through his book series, developing innovative software solutions for their clients and watching classic sci-fi movies in his free time.

To learn programming from an expert, look out for more of Nathan's books in store and online.