C PROGRAMMING for Beginners

CRASH COURSE

MARTIN LAREDO

C Programming for Beginners

Crash Course – Ultimate Edition

Table of Contents

Introduction

Congratulations on downloading *C Programming for Beginners: Crash Course – Ultimate Edition* and thank you for doing so.

The following chapters will discuss some of the things that you need to know in order to get started with working in the C programming language. This is one of the first coding languages that worked to make things easier for people to use. Many of the programming languages that came before this were too complicated and only those really advanced in computers were able to figure it out. The C language changed all this and made it easier for anyone to learn how to code and it is now the basis for many of our modern languages.

This guidebook is going to take some time to look over the basics of the C language and how you can learn how to use it on your own. We will discuss some of the history that comes with this programming language as well as the basics in functions, variables, and keywords before moving on to comparisons, such as the if and if else statements inside of this language.

When you are done with this book, you will have experience writing some of your own code, with some of the examples that are inside, and the confidence to go out there and make some new programs. When you are ready to learn one o the best programming languages out there, and one of the first that made programming easier for everyone, make sure to read through this guidebook and learn how the C programming language can work for you!

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!

Chapter 1: An Introduction to the C Programming Language

When it comes to learning a new programming language, there may seem to be a lot of parts that you need to keep track of. You may worry as a beginner that it is too hard to put together a code on your own and that you need years of experience. Even if you have worked in coding in the past, you may worry about how difficult it will be to learn something new and how different it is from your chosen computer language. But coding can be simple to learn, even when you are brand new to the process.

In this book, I am going to take some time to show you how to use the C programming languages, one of the best options out there for all the options and power that come with learning how to do your own coding. I will show you some of the jargon that comes with learning how to program and walk you through some of the steps that are needed to end this book with writing your own code.

How Did C Programming Begin?

The C programming language started in 1972 with the help of Dennis Ritchie. At the time, Ritchie was a computer scientist at the AT&T Bell Laboratories and he needed to develop a few programs for his own personal use. He was trying to make the process of computing as simple as he could and he realized that many of the current languages were too complex for what he wanted and for what other people were able to use.

The goal with this new language was to help create a programming language that was efficient, but also one that could help the community form a fellowship. They knew that to keep things working smoothly you couldn't just have information put into the terminal of the code, but you needed to have some post programming communication there as well.

The result of all this work was the C programming language. This is a language that is general purpose as well as structured and it is known as a procedural oriented programming language. It is not a language that is designed for a specific application, but it is well suited for those who need a scientific or business application. There are many features inside, such as micros, looping statements, and control structures that are required for the applications.

There are many different features that you are going to enjoy as a user of the C language including:

- Interactivity: this was one of the first of its kind to interact with the user and become easier to use, even without a lot of experience.
- Reliability: the C language is really reliable. You will be able to write out your codes and not worry about them crashing or having issues down the line.
- Efficiency and effectiveness: if you want a programming language that is effective and gets the job done, none will work the same as the C language.
- Flexibility: this language is really flexible. There are many different programs that you are able to work with inside this language and while it is mostly used for business purposes, you would be able to use it for any of your own needs.
- Portability: the C language can go where you go, working on a variety of different platforms whether you are on a Windows computer, Mac, Linux operating system, or something else.

While there have been many programming languages developed over the years and the changes made, the C language is still a great one to learn how to use. Many programmers still use the C language and many of the newer programming languages will use C as their foundation. If you want to learn a language that will make other coding languages easier to learn down the road, the C language is the best option for you.

Getting Your IDE

Before you are able to use your C language, you need to make sure that you download a good IDE to go with it. The IDE, or the Integrated Development Environment is basically the area where you will write, link, edit, compile, and run your code. Your C language is basically not going to work unless you have an IDE in place to help you out. With the right one, you will be able to do all the tasks above plus work with a variety of graphical tools, create your own complex programs, and even do some debugging.

Luckily, the IDE for most of these programming languages are free to download and use online so it won't cost you a lot of money to bring them up. For this book, I am going to use the Code Blocks IDE to run the C language, but there are different options to help you out depending on what you would like to do with your code.

The Code Blocks IDE is one of the best because it will have all of the features you are going to need. You won't have to go online and find a compiler or other parts to help get the code written because it is all connected as one. You will be able to download Cod Blocks by visiting the following website http://www.codeblocks.org

With the right IDE in place, you will be able to use the C programming language and get some of your code written in no time. It doesn't matter if you are used to working in programming or you are just beginning, I will walk you through some of the steps that you should take in order to create great codes and great projects that you can do with the C language.

Chapter 2: The Basics to Writing Your First Project with the C Language

Working with the C language is pretty simple. You can write a code that just has one line or you can go on and write a nice long one that is going to be more complex and could run a game or some other process. While you can get more complex later on, I am going to look at some of the basics of writing inside of the C language to help you to get comfortable.

Writing a Dummy Code

First, let's take a look at writing out your dummy code. This isn't going to bring you an output like some of the other codes that we will discuss later on, but it will help you to get the hang of creating and saving a code in the C language so you are ready for the things we do later on.

So to start, you need to go into the Code Blocks IDE, or whichever IDE that you chose, and click on the New Button. You will want to open up an Empty File. Now you will need to type in a code into the editor so that you are able to create a code to save. I am going to keep this simple and just type in one line like the following:

```
main() {}
```

Now you will need to save the source file and you can do this by clicking on the Save button. You can either let this save on the default of your computer or you can choose the folder that you would like all of these code files to be saved on your computer; the choice is up to you, just remember where you are saving all of these so if you need to find them later.

Make sure to name the file something that you will be able to remember later on, or you will get confused as you start to add on more of these files over time. I am going to call this file "dummy.c". once you have saved this, the source code file is created and it has been saved on your computer. Now you will click on the Build button.

You will notice that the code is not going to compile. What you are going to see is the minimum of the C program, which is also called the dummy. All of your codes in C need to have the main function because this is where the execution of the program is going to start; you will just need to put the main function inside of the curly brackets.

Since this is a dummy source code and one where we are just experimenting a little bit, I didn’t have us put any code into the curly brackets. When you try to run this option, you will not get an output because nothing was placed inside. We will be able to add in different things later on and create an output based on what is inside the code. You may see a compiler warning when entering the dummy code, but this is not critical. You will just click on your Run button and then find that it is not able to give you any output.

Congratulations! You have just written your very first code using the C programming language. If you didn’t get any output, you did the code right. This is just the basic form of writing a code in the C language and there is so much more that you are able to add into the code. Some of the other basic parts that you can add into your code includes:

Structure
Variables and values
Operators
Functions
Keywords

I am going to take you through how some of these work so you can learn how to make the code shine the way that you would like.

Keywords

The keywords are important to learn about in the C language because they help you to accomplish the basic tasks that you want when writing out a code. There are 44 words that are reserved as keywords in the C language and you should only use them as such inside your code. For the most part, you may find that most of them are only used on occasion so you will become familiar with the rest pretty quickly. I will show you some of these keywords as we progress through writing some of the codes later on in this book.

Functions:

Functions are really important in this language and many of them are held inside the library that are found in the C language. The function is going to knit together the code so that it is able to make the program that you want. When you want to make or use a function inside of your own program, you will need to incorporate in a header file, which is going to define the function. There is so much that you are able to do with your functions and I will give some examples of how powerful this can be later on.

Operators

Operators are gong to be used inside of your code in order to manipulate the data that is inside the program. There are several types of operators and each of them are going to work in a different manner depending on how you use them. Some of the most common operators used inside of the C language include:

- Mathematical: these would include any of the signs that you would need in order to add, subtract, divide, or multiply information together in the code.
- Comparison: these operators allow you to compare different values or pieces of information to each other inside of the code to determine if the statement is true or false.
- Assignment: the assignment operators make it easier to assign a value to your variable inside the code.

Variables and Values

The variables and values are going to be similar to each other, but there are a few differences. The values are going to contain numeric values and characters and you can make them contain anything that you would like. For example, the numeric values can be fractions and decimal points or really small or really big numbers. Or with the characters, you could have one letter or a whole sentence or more depending on what you would like the code to do.

With variables, you are working with the containers for a value. You can change up the contents as much as you would like and they can vary. The variables are basically going to hold the same values that you are going to place directly into your program later on.

The Structure

I am also going to talk about the structure that goes with the C language. This structure is going to help you to write out the codes the proper way and can control the way that the program flows. To do this, the C language is going to use the preprocessor directives. The first function that will be ran inside of the C language is the main function, like what we used in our first example above. This main function is a big requirement for all of the programs that you want to use because without it, your program will not compile or run at all.

Next, you will need the curly braces or the brackets to help enclose the contents of your functions. Inside the brackets, you will be able to include statements, which are basically sentences that will contain math, comparisons, keywords, functions, and more to help the compiler to understand what you want executed inside of the code.

Another thing that you may want to use at times inside of the code is the comment. The comments are basically little footnotes that you can write for the other programmers, and sometimes even for yourself, when looking back inside of the code. The comments can tell someone else what is supposed to happen at certain places in the code, but when they are placed properly, the compiler will just read through and not execute the comment.

Comments can be really helpful for making sure that everyone knows what is going on in the code or explaining what you would like to have happen at a certain place in the code. The compiler is not going to read these comments so you will be able to write out as many as you would like inside the code.

Writing Another Code

Now that I have had some time to explain a few of the basic parts of code writing, let's take a look at how this would work by bringing out our dummy.c program from earlier. Just open up the code from wherever you stored it before and I am going to make the main function be defined as an integer function. This basically means that it is going to return an integer value onto the operating system. We will need to do some editing in order to make this happen.

Inside of your editor, you will need to add in the keyword "int" before your "main" part to ensure that you are getting the integer output that you would like. Make sure that you place a space between both of these keywords to help the compiler to read through both of them. So to start, type out the following code:

```
int main()
{
}
```

You will notice that the code is a bit different than we originally wrote out with the first code, but putting the curly brackets in this manner is what most veteran programmers prefer to use. Now it is time to add in a statement to this main function so that it will actually show you an output. I will keep it simple and just add in the number three.

You will first need to type in the "return" and then the number three. I will write out the example of the syntax that you would use to make this happen:

```
int main()
{
                    return(3);
}
```

Make sure to add in the semicolon after the statement. Save the file and then click on the Build button. As long as you type in the code like I wrote above, you shouldn't have any issues with errors or messages coming up for you. Click on the Run Button.

When using a Linux or Mac system, it is possible that you won't see any output other than the build log, and it will say that the program terminated with a status of zero. On a PC computer, the terminal window is going to show you the return value 3.

Any time that you want to add in an output to your dummy program, you need to bring in the output function. The keywords in the C language aren't going to output anything because they are just basic vocabulary, such as the words int and return. You can do this by using the "puts" command inside the program.

Let's take a look at how this is gong to work when you write out some code. Make sure that the function of the "puts" is inside of parenthesis and that you place n a string of text between a double quote to make it work properly. Here is the example I am going to use:

```
int main()
{
        puts("I am the King of the C programming world");
        return 3;
}
```

When you save this source code and then click on Build, you should see a warning come up on your computer. Even if you don't see this, you need to realize that you have another step to do at this point. Before the puts function is going to work, it needs to have a definition inside, or you will find that the compiler is confused. The definition of your puts will be in the I/O header file and you need to place this into the source code with the help of the preprocessor directive.

Here is an example of how that would look inside of your code to keep things organized and to avoid errors.

```
#include<stdio.h>

int main()
{
        puts("I am the King of the C programming world");
        return 3;
}
```

This version includes the preprocessor directive along with the definition for the puts function. You can save the file and then click on the Build and Run buttons that are at the top of the editor. If everything is typed properly, you can avoid errors and in the output terminal window, the statement that you wrote out for the puts function as well as the value 3 will show up on the screen.

Learning some of the basics of writing a code in the C language can make it easier to understand what is going on. I showed you a few of the options that you have when working inside of the C language, but there is still so much more that you are able to do. Take some time to get familiar with how these codes work, play around with the IDE and the compiler, and get comfortable before moving on and learning some more complex options when writing your own code.

Chapter 3: Language Comparisons Inside the C Programming Language

In most cases, when you write out a code, the program will go through and execute the statements as they show up inside the code, going one line after the other. But there are times when you will want to make changes to the flow of your program. The if statements are one of the easiest ways to change up the order of your program flow and makes it so that the program is able to make decisions for you rather than just reading through the code line by line.

In this chapter, we will spend some time looking at the if keyword and learn how different comparisons are made inside of the C language so that the compiler is able to handle multiple decisions based on what you place inside the code and what the user inputs into the system.

Working with Comparisons

The first part that I will discuss is how to make the comparisons work inside of the C language. When doing comparisons, there will be two expressions or values that the program will evaluate. To make these evaluations, you can use some of the comparison operators that are found in this language. Some common operators for comparisons include:

- (!=): this one means does not equal
- (>=): this one means greater than or equal to.
- (<=): this one means less than or equal to.
- (<): this one means lesser than.
- (>): this one means great than.
- (==): this one means equal to.

Now that we have learned a few of the most common comparison operators, it is time to take a look at how this would work within a code with the following example.

```
#include <stdio.h>
int main();
{
        int a;

        printf("Type an integer:");
        scaf(%d",&a);
        printf("You typed %d. \n" ,a);
        if(a>10)
        printf("%d is greater than \n" ,a);

        return(0);
}
```

In this example, your user is going to be prompted to type in their integer value. The value that is displayed at line 9 and then when you get to line 10, the if statement is going to evaluate that value. If it determines that the value is above 10, then the 11th line will be executed. But if the value is lower than 10, then the 11th line is skipped. Take some time to build up and then run this code to get it set up.

Once you have done that, type in any value that you would like. I am going to use the number 45. When you type this number into the code, it is going to work and the compiler is going to display the printf statement that is on line 11. But if I chose a number like 5, the 11th line and the statement that goes with it will be skipped and the program will terminate.

If Statements

I talked a bit about how the if statement will work inside the code above. For the most part, the if statement is going to be formatted a bit different than some of the other statements we will work on. It is going to be split out amongst at least two lines. The first line will have the if condition set apart in parenthesis. If it evaluates the condition and it is found to be true, then the following line will be executed. It is basically just one statement that has been divided between two lines, so the semicolon will be with the second part of the statement instead of after the first part.

Now, you could write out the if statement into just one line if you would like, but this is not common programming etiquette and usually it make the code easier to read when they are separated out into two or more lines and enclosed in brackets.

You can also choose to have more than one if statement in the code if you would. This allows you to have a few things happen inside the code based on the conditions that you set out and what input the user gives to you. Here is a good example that you can try out inside your program as well:

```
#include <stdio.h>

int main();
{
        int a;

        printf("Type an integer: ");
        scanf("%d", &a);
        printf*:You typed %d. \n" ,a);
        if(a > 10)
        {
        printf("%d is greater than \n" ,a);
        }
        If(a < 10)
        {
        Printf("%d is lesser than \n" ,a);
        }

        return(0);
}
```

With this code, we can see that there are two if comparisons right next to each other.

Both of them are going to have their own set of brackets around them and two different conditions are going to be evaluated; one that is looking at greater than and one that is looking at less than. Take a moment to build this code and run it.

After you run the code, you will see that the necessary output is going to be displayed for values that are either greater than or lesser than 10 based on the input that you put in. Notice with this option that there isn't going to be an output that shows up when you type in the number 10, only when you have numbers that are bigger than or less than 10. You can change this up in line 16 so that it says "less than or equal to 10" in order to change this issue and get an output when you use the number 10.

Else Statements

While the if statements can help to add some comparisons to the C language, you can also use the else statement to help you out. These are great for the either or decisions that you want to create. Take a look at the code below to see how the else statements can work inside your code:

```
#include <stdio.h>

int main()
{
        int a;

        printf("Type an integer: ");
        scanf("%d", &a);
        print("You typed %d. \n" ,a);
        if(a > 10)
        {
        printf("%d is greater than \n" ,a);
        }
        else
        {
        Printf("%d is less than or equal to 10. \n" ,a);
        }

        return(0);
        }
```

The code above is basically a rewrite of the codes we used before, but we added in the else statement. In line ten, we used the if condition in order to make the evaluation. This statement is going to be executed if that condition is evaluated as true. If the condition is evaluated and seen as false, the statement that is with the else part of the code will be executed. Take a moment to build and run this code and try out a few inputs to see how it works.

Note in this code that I didn’t use a semi-colon, but instead of a curly bracket. You could write it all out without using the brackets since there is only one statement, but this is a great way to write it out to make sure that it all stays together and it is good programming practice.

Else If Statements

There are times when you will have at least three, if not more, conditions that you want to use inside of your code. In these instances, you would need something a bit different than the if and the else statements. This is when we will use at the structure for the else if statements. Let’s take a look at the following code to see how the else if statements would work.

```
#include <stdio.h>

int main()
{
        Int a;

        Printf(“Type an integer: “);
        Scanf(“%d”, &a);
        Printf(“You typed %d. \n” ,a);
        If(a > 10)
        {
        Printf("%d is greater than \n" ,a);
        }
        Else if(a < 10)
        {
        Printf)"%d is less than or equal to 10. \n" ,a);
        }
        Else
        {
        Printf("%d is 10, \n" ,a);
        }
        Return(0);
}
```

With this structure, you can start with the if statement that is at line ten. If this condition is true, the statement that is associated with it will be executed. If the statement is evaluated as false, it is going to move on to the else if statement that is examined at line 14. If this condition ends up being evaluated as true, the statement will be executed and all the rest will be skipped over. At the very end of this is the else statement. If neither of the two conditions above it are met, this is the one that will be executed.

The nice thing about this is that you can add in as many of these else if statements as you would like. This makes it easier for the system to evaluate a complex situation based on the conditions that you set and the input that your user places into the system.

Switch Statements

Inside of the C language, you are able to use the if and else conditions to deal with complex decisions. You can add in as many of these as you would like to the system t deal with this, but the more conditions that come into play in your code, the uglier the code can become and the harder it is to write it all out. Instead of dealing with this, you can use the switch case structure, which will also help you to make some more decisions in the C language without all the mess. Let's look at the following code to see how the switch statements look inside a code and then how it can handle a variety of decisions at the same time:

```
#include <stdio.h>

Int main()
{
        Char a;

        Printf("Your choice (1,2,3): ")'
        Scanf("%c", &a);

        Switch(a)
        {
        Case '1':
        Puts("Excellent choice!");
        Break;
        Case '2':
        Puts("This is the most common choice:");
        Break
        Case '3':
        Puts("I question your judgment.");
        Break:
        Default:
        Puts("That's not a valid choice.");

        Return(0);
}
```

The majority of this code is put in the switch/case structure. It is going to start out on line 10 as a switch statement and then it has a series of the case statements, each of which will have their own statements (which you will be able to change up in any way that you would like based on the code that you are writing). The switch case also has its own statements and then a closing curly bracket. All of these are important elements that come with the switch/case structure.

You may have noticed that it has the break keyword inside of it. So there are four keywords that are found inside this structure including break, default, case, and switch. Place this inside of your compiler and then run it a few times, picking out the different options to see what the output is and if you placed all of the stuff inside the code correctly.

So how does all of this work? The switch statement is going to handle a single value rather than a whole comparison. It can hold onto a mathematical equation, but the result needs to end up being just a single value. The value, which is specified by the switch, is then going to be compared to the values of each of the case statements. If it finds that the comparison is true, the statements that belong to that case statement will be flagged and executed. The ones that are not seen as true will be skipped.

For example, if the user puts in the number 1 for line 12, then the puts statement that is in line 13 will be the one that is executed and the rest are skipped. But if they pick the number 2, you will see that the first case is skipped and it will move over to execute the second case and so on until it finds the one that is true.

If there is a reason that the user puts in a value that doesn’t match any of the case conditions, your code will go to the default condition. In this example, if the user puts in the number 4, it would skip to the default and execute the code that comes with it.

You are able to use this for any code that allows the user to put in some input like a menu function. This gives them some choices and you are able to teach the code how to interact based on the answers, or the input, that your user gives. As long as they give an answer that is allowed inside of your switch statement, they are going to see the right answers that you put in!

Chapter 4: Using Loops Inside of the C Language

Loops can be a helpful thing to add into your code in the C language. These allow you to create lists of things or even graphs and charts inside of your code. They are nice because with just a few lines of code, you could get a huge chart done, rather than trying to type out each and every line that you need along the way. For example, if you wanted to make a chart that went from 1 to 100, it wouldn't be much fun to write it all out for each line, but with the right loop statement, you would just need a few lines to make this happen.

Loops are another way that you are able to do comparisons inside of your code in the C language and there are several options for looping that can really add some power to your code. The while loop the nested loop, and the do/while loop are the most common types of these that you are able to use, but I will also spend some time talking about the for keyword in this chapter.

The While Loop

A loop is going to be a control program execution that is very repetitive. You will be able to specify the condition that makes that loop repeat, as well as a set of statements that you want to repeat in the code, and a way for the loop to stop. If you forget to add in the part about stopping the loop, the program is basically going to keep going on forever and freezing up the computer so make sure that this is added in.

Here is an example of a code that will use the while loop in order to have the code count from 1 to 10.

```
#include <stdio.h>

Int main()
{
        Int x;
        x = 1;
        while(x <= 10)
        {
        Printf("%d\n" ,x);
        x++;
        }

        Return(0);
}
```

The condition for looping is shown at line 8 and it reads that when the value of your variable is less than or equal to the number ten, so as long as your condition ends up being true, the statements inside the loop will continue to repeat. Line 11 shows that the value of x is going to be incremental so it will go up by one each time that the loop is executed, until the value of x is bigger than 10. When you build and then run this code, you will be able to see a display of the numbers 1 to 10.

You will be able to make changes to this to get as many while loops as you would like. For example, say that you would like to have the code count to twenty. You would just need to go to the part with the while statement and change out the number that you want to use. You can also change up how many increments that you would like to go up by. The code above is set so that x = 1 so it just goes up by 1, but you could change this variable around to be anything that you would like, such as counting by 2s or 5s to get the right answers that you want.

Do/While Loop

Another option that you have with doing loops is the do/while loop. This is similar to the while loop, but it is almost like an upside down one. With the while loop, the code is going to see if the conditions are true and if they are, it will then execute. With the do/while loop, the code will execute once, and then check to see if the conditions are met before running again. If you want to make sure that the statement runs at least one time, the do/while loop is the best option for you to use. Let’s take a look at the following code to see how the do/while code will work.

```
#include <stdio.h>

Int main()
{
        Char ch:
        Ch = 'A';

        Do
        {
        Putchar(ch);
        Ch++;
        }
        While (ch != 'z');

        Putchar('\n\);

        Return(0);
}
```

You will notice that the do/while loop is going to start with line 9 with the statement "do". this one doesn't have a condition because the statements are going to be executed one right after the other. Then you can see that the while condition is going to appear at your 14th line. The loop is going to rely on the ch variable as the condition and it is first initialized at line 7 before being manipulated at line 12. Take a moment to build and then run this code in your compiler.

In this one, the 'z' won't be printed because it is considered an exit condition. If you would like to see the 'z' in your code, you will need to save the exit condition to 'z+1' at the end of your code. Now with this one, you are basically creating an endless loop because it has the plus one. If you built and tried to run this one, you would end up in an endless loop and would need to press Ctrl + C in order to get out of the program.

There are times and places when you would want to use an endless loop, but as a beginner, you will most likely come across this unintentionally. There are ways to fix the endless loops by adding in a break statement to your code. Here we will insert the break statement right at the 8th line of the code like in the following:

```
#include <stdio.h>

        Int main()
        {
        While(1)
        {
        Printf("I'm endlessly looping!");
        Break;
        }

        Return(0);
}
```

Now save, build, and then run the code. Since the break statement is in there, the loop is only going to spin through once, rather than having it go through and keep looping all of the time. You can even use this break code in order to allow the loop to break out prematurely if you would like.

Basically, the while loop and the do/while loop are very similar. With the while loop, you will have the code look to see if the conditions are met and if this is true, the statements or the loop will execute. On the other hand, the do/while loop is going to execute at least once, doing the execution and then determining whether the conditions are true before looping again and again. You could use them the same way, but if you want to make sure that the loop happens at least once, you would go with the do/while loop.

The For Keyword

The for keyword is a great one to use when you want to create a loop inside the code.

Unlike using the while loop, the for loop can be done with just one line, which does make it a bit more cryptic to work with, but it is still a more popular and traditional loop type to work with. A good example of this includes the following:

```
#include <stdio.h>

Int main()
{
        Int x;

        For(x=0; x<10; x++)
        Printf("%d\n" ,x);

        Return(0);
}
```

The first part that you see is the initialization, the statement is x=0, which is the assignment that is made at the start of your loop. The second thing inside this code is the looping condition. The loop is going to continue to repeat until the condition is no longer true, which in this case is until the x equals 10. Finally, you will notice the iteration statement. Each time that your loop repeats itself, this particular statement is executed.

The semi-colon is not going to follow the parenthesis of the for statement, but instead it is going to follow the single looping statement, which is inside the curly brackets on line 8. Take a moment to build and then run this code. As this is set up, the code is going to start at 0 and go up to 9, and then the loop stops as soon as this gets to 10 as the conditions are no longer true at that point. In this program, unless you specify differently, the program is going to start the counting at 0 rather than 1.

Since most of the time you will want to start the counting at 1 since this is what most people are used to, you would go to line 8 and change the x to x+1. This isn't going to make any changes to the value of x, but it does make a difference to the output. Make sure to save, build, and then run this code to see what it can give you.

The Nested Loop

The for loop is really helpful for cleaning up some of the code writing and it can help you out when working on what is known as the nested loop. The nested loop may sound complicated in the beginning, but it is easier to use than the while loop. Let’s take a look at how to work on the nested loop by looking at the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

Int main()
{
        Int column, r;
        Srand((unsigned)time(NULL));        /*Seed Randomizer*/

        For(column=0; column<10; column++)
        {
        r = rand();
        printf(“%2d\t” ,r % 100);
        }
        Putchar(‘\n’);

        Return(0);
}
```

The code that I put above is one that will output ten randomly selected numbers. Take a moment to build and then run this code and then you will see that from your output, there is a row of ten values that are randomly selected. The spacing that is between the values is going to be attained by making use of the ‘\t’ character that we used at line 14.

This is just one of the examples of what you are able to do when using the nested loops. You can also use it to make a grid of numbers 1 to 100, or more, and to add in a variety of different items that you need inside of your graph. It is quicker to do than writing out each of the parts on their own and can really make for a nice end product when you are done.

Chapter 5: The Variables in the C Programming Language

As I mentioned a bit before, a variable is going to be the container for different data type sin your code. The variable, or the container, will be able to hold a floating point character, an integer, or some other value. In this chapter, I am going to explain a bit more about the variables and how you can use them, and how they work, inside of your C language code.

So how do you distinguish between a variable and a value. A value can be almost anything inside of the code, such as an integer, a character, or something else that can be defined in the code. On the other hand, the variable is going to be the container that holds these data types together. Usually they have to have something that ties them together, but they don't need to be identical. For example, you could have a variable that contains school supplies such as pencils, pens, paper, chalk, notebooks, and so on. These are similar in one way, and they make sense to go together, but they are not identical.

When you declare a variable in the C language, you will learn about the type of variable and the name of the variable. Let's take a look at how you would declare an integer variable to start:

```
#include <stdio.h>

Int main()
{
        Int a;

        Printf("The value of a is %d\n" ,a);

        Return(0);
}
```

In this option, the integer variable is “int” and the name of it is a. the variable in the above code is never assigned a value, but even though it is never given a value, you will still be able to use it inside the code, just like it is shown in the 7th line. Build and run the program really quick. On your side, you may see that a random value comes up. Unlike some of the other languages you will come across, the C language is not going to initialize the variables as they are declared. Internally, the program is going to allocate a chunk of the memory in order to store the information for that variable.

In this case, the chunk hasn’t been initialized and yet it is not set to zero. It just is placed in some location in your memory. Whatever value is already placed there, it is going to be absorbed right away by the variable and you will see this s the output on your screen. The idea behind this is that you need to initialize the variable before you try to use them. A good way to fix this issue is with the following code:

```
#include <stdio.h>

Int main()
{
        Int a;

        a = 65;
        printf(“The value of a is %d\n” ,a);

        return(0);
}
```

Save these changes and then build and run the program. Now the output of your code is more predictable because you know the value of the variable. In addition to knowing the name and the value of the variable, you can also learn a few other things about your variable in this language. One of these is the sizeof operator.

Sizeof is one of the keywords in the C language, but it is also considered an operator. What the sizeof keyword does is return how many bytes of storage are used for that variable. This may not seem too important, but when you are working with the C language, it can come into play and you will need to know this information at times. Here is an example of how you can use the sizeof keyword in your code:

```
#include <stdio.h>

Int main()
{
        Int a;

        Printf("An int variable occupies %lu bytes of storage\n", sizeof(a));

        Return(0);
}
```

In this example, the variable a is not initialized, but since it is not used, this is not a big issue. The sizeof operator can be find inside the printf statement and it is simply going to evaluable the variable a as an int variable to return how much storage on the computer is being occupied. The output that you will get is a long and unassigned integer value. You can change up the information that you would like to receive, such as looking up how much space a character will take up and not just the variable, by changing up some of the code to reflect this.

Another thing that you will be able to find out about your variable is the location of the memory. This is going to tell you specifically where the data of the variable is located. To fetch the information about the memory location, you will need the ampersand operator. Here is a good example of the code that you would use to make this happen:

```
#include <stdio.h>
Int main()
{
        Int a;
        Char b;
        Float c;
        Puts(“memory location:”);
        Printf(“A is at %p\n”, &a);
        Printf(“B is at %p\n”, &b);
        Printf(“C is at %p\n” , &c);

        Return(0);
}
```

There are three different variables that are declared inside this code; the floating point value, a character, and an integer. These variables are not initialized inside this code because they aren’t being used, but the program will still allocate some space for them at different locations inside of the memory. In order access these locations, the ampersand is prefixed to the name of the variable. You will need to use the %p placeholder inside of the printf statement in order to get the results that you want.

The output values that you will receive will vary based on the machine that you are using. Even the formats can change based on the operating system that you are using. The computer will be in charge of assigning a location on the memory and it sometimes depends on how much memory is available on your system or where the system decides to place it. Learning where the information is stored on your computer may seem a bit trivial at this point, but it can make it easier to find things and work on your code later on.

Working with variables can help you to store information in your code properly. You are able to assign any value to your variables that you would like, you just need to be careful that you are initializing it inside of the code that you are writing and that it makes sense with all the other data types that are being stored inside of the variable.

Chapter 6: The Basics of Functions When Writing Your Code

Like many other programming languages, the C language contains a library. As a beginner in coding, you will find that this library is your best friend. It allows you to go through and fins some functions of the code, the bones of the code, that helps you to write out what you want to happen. It can give commands to the compiler, helps you to set up the code, and so much more.

The C language library has many functions that you are able to utilize and these functions are going to help you accomplish quite a few things. With that being said, they aren't able to do everything and there are times when you will need to make up some of your own functions in this language. In fact, many programmers who have been working in this language for some time have created many of their own functions to help the code work properly.

In this chapter, I will spend some time talking about the function, including how to setup a function, how to prototype it, how to call or access any function that you would like, and even how to bail out of your function early if you need. In the C language, the function is going to have a type, a name, and arguments that are held in the parenthesis. The name is going to be unique and should be used to identify the function that you are using. The arguments are zero, one, or more values that will be passed through the function and is known as the input.

Below I have the alpha function, which isn't going to have an input or an output; this is something that is valid inside of the C language. This type is going to be known as the void function and will look like the following:

```
void alpha(void)
```

The Void is the variable type, which basically tells the compile that you don't have a variable type. The function count is going to be the integer function and it is responsible for generating or returning some kind of integer value. It will need some more input than above, and therefore it doesn't have any arguments. It would look like this:

```
int count(void)
```

Another thing that you can do with functions is the function hangup, which is going to return no values, so it is also a void function. However, this one is slightly different because it does accept a single character as the input and you can choose to have the character as an argument. In the C language, it is going to be referenced with the "ch" variable. Before you are able to use your function though, regardless of the type that you are using, you need to introduce it to the compiler and allow the compiler to check the arguments to ensure that everything is being used properly.

When you want to call up the function, you will call it by the name and then use a parenthesis. When the function is called, the control passes on to the statements of the function. Once the function is all over, the flow will continue with the statement after the function call. Let's take a quick look at how the function can work in a code. The following code will contain two functions to help out with this:

```
#include <stdio.h>

Void blorf(void);

Int main()
{
        Puts("the main () function always runs first");
        Blorf();
        Puts("thanks, blorf()");

        Return(0);
}
```

The main function is again the one that you need for all codes in the C language. You will also notice that there is a blorf function, which is new to the system. In line three, you are prototyping the function and informing the compiler of the function argument and types. The compiler can then confirm that you are using the function the correct way inside the code by checking out the prototype. And sense the prototype is a statement, it is going to end just like the other statements with a semi-colon.

Then we move on to line 8 which is going to call the function. Since this function doesn't have an argument, the parenthesis is going to remain empty. The function itself is not going to be found until line 14. This will effectively repeat the prototype without the semicolon. The statements that belong to this particular function are going to be inside the curly brackets. Now, take a few minutes to build and then run this code to see how it works.

Now say that you want to call up the function again. This is simple to do because you would just need to duplicate line 8 over to line 9 and it would repeat itself again. So the new code would look like this:

```
#include <stdio.h>

Void blorf(void);

Int main()
{
        Puts("the main () function always runs first");
        Blorf();
        Blorf();
        Puts("thanks, blorf()");

        Return(0);
}
```

When you save and run the code again, you will see that this output will appear twice.

The prototype inside your code is going to work similar to a definition to the function, basically describing to the compiler how it is going to be used before the function appears in the code. There are times that you won't need to specify the prototype, but this is only going to work when you write out the function before it is used. A good example of how this works includes:

```
#include <stdio.h>

Void soup(void);
{
        Puts("Pea green soup!");
}

Int main()
{
        Printf("For breakfast I had");
        Soup();
        Printf("For lunch I had");
        Soup();

        Return(0);
}
```

The soup function is going to appear before the main function in this code. This is just fine because the compiler is still going to see them inside the function and it is always the first function that is going to be executed inside the program. What is missing in this one is the prototype of the soup function. Take a moment to build and run this code as well.

Now I will show you another example that you are able to use that will allow you to use the return keyword in order to bail out of your function early if you would like. This could be used when dealing with a loop or some other thing that is continuing inside of your code that you want to stop early. Look for the return at line 26 to see how this works:

```
#include <stdio.h>
```

```
Void cheers(void);

Int main()
{
        Int x;

        x=21
        puts(“Everyone gets free dinner!”);
        cheers();
        puts(“Everyone gets free dessert!”)
        cheers();
        printf(“%d\n”, x);
        puts(Everyone pays higher taxes!”);
        return(0);
}

Void cheers(void);
{
        Int x:

        For(x=0;x<3;x++)
        Printf(“Huzzah!”);
        Return;
        Putchar(‘\n’);
}
```

In this one, you don’t have to go through and specify a value because the cheers function is considered a void function, meaning it isn’t going to return anything. In the main function, the return needs to specify a value at line 17, like it does in this example, because main is considered an int function. Save the changes of this one and then build and run your code.

The output on this one can be a bit messy because your putchar statement is skipped. However, the code is never executed because of the return statement that is placed right above it. It is more common to see a function return early based on a condition, which is usually going to be shown with a loop or an if statement. Otherwise, if you see the return function, this is the spot where your function is going to bail out and the control will go back to calling the function.

There are many was that you will be able to use functions inside of your code. They are going to help the compiler to see what is going on inside the code, as long as you define them properly, and while there are many of them already available in the library for the C language, you can also have the ability to create some of your own. Keep in mind that they need to be defined before using them in this language, or the compiler is going to get confused as to what you want to do and will often send up an error message. Overall, these functions give you a lot of freedom in what you are able to create inside your code.

Conclusion

Thank for making it through to the end C Programming for Beginners: Crash Course – Ultimate Edition, 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 with writing out some of your own code. You will be able to try out a few of the codes that we offered inside of this guidebook or use the tips and tricks that we discussed in order to write the code to fit your own needs.

As a beginner, you may be worried that writing code is going to be too complicated, but the C programming language makes this one really easy to work with. It is the basis of many of your newer programming languages as well so you will be able to use these tips to put them all together and learn other coding languages in the future so getting started doesn't have to seem so intimidating.

Inside this guidebook, you will learn just how simple the C programming language can be. We will talk about many of the different parts that come with this language including the history of the C language, some of the basic parts that are found in the codes of this language, how to work with comparisons in the C language, and even how to work with the different types of loops and so much more. When you are done, you will be able to use the C language in order to create some of your basic codes and you will have the confidence to keep working and creating some of your own programs as well.

When you are ready to get into the world of coding and want to learn how to make some of your own codes and projects, take a look through this guidebook and learn how the C programming language will be able to make this all a possibility for you.

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

Description

The C programming languages may have been around for awhile, but it is one of the best that you can use. This language was one of the first developed that made it easier for people to learn programming, with more efficiency and a good readability, compared to some of the other programming languages of the past.

In this guidebook, you will learn some of the basics that you need to know in order to get started with the C programming language. Whether you are interested in getting started with a new coding language to add to your arsenal or you are just starting out for the first time, this guidebook will help you to get through some of your first codes to get the results that you want.

Inside this guidebook you will learn the following about the C programming language.

- The beginnings of how the C language started
- The basics to writing out your first project with this language
- How language comparisons work in the C language
- Using Loops to save time in your code.
- How variables work with this programming language.
- Some of the basics of functions when working on your code.

When you are ready to learn one of the best programming languages out there or you want to get some of the fundamentals of other programming languages, make sure to read through this book and learn more about the C language and how it can make a difference in your projects!