C Programming FAQ

This is a compilation of some frequently asked questions relating to my course, *C Programming For Beginners*.

Huw Collingbourne
(Course instructor)

Using CodeLite

Although it is possible to write and compile C programs using many different editors and IDEs including *NetBeans*, *Eclipse*, *Visual Studio*, *Code Blocks* (and many others), I have used the *CodeLite* IDE when for creating both the source code archive and demonstrating the use of C code in the videos. I therefore strongly recommend that (unless you are already intimately familiar with another IDE and know how to import and build C code with it) you use *CodeLite* when following the course.

PLEASE NOTE: I am not able to provide technical support for *CodeLite* or other IDEs. If you need support on *CodeLite* I recommend that you seek assistance on the *CodeLite* wiki here:

http://codelite.org/LiteEditor/Documentation

And the *CodeLite* forum here:

http://forums.codelite.org/

If you experience insurmountable problems installing or running *CodeLite*, you may consider using another IDE or editor as explained later in this FAQ.

Can't Compile C on OS X

Question: I've installed a C IDE but I still can't compile any programs. What's wrong?

Answer: You need to install Apple's free Xcode IDE and its command line tools in order to install a C compiler on OS X. This is explained in the lecture called '*Install a C compiler on a Mac*' in Step 1 of this course. You can also download the relevant compiler by logging onto the Apple Xcode site, logging in and clicking the link to *Additional Tools*: https://developer.apple.com/xcode/download/ then, from the download page, download the *Command Line Tools*.

Creating a C project in CodeLite

Question: When I try to create a new project, the *CodeLite* version I have has a "new project wizard" that asks me which project template to pick from. The available options are GUI, Console, User Template, Unit Test++, Library, and Others. I'm a bit confused on which one to pick. Any guidance here?

Answer:

Try this.

- Select: *File* | *New Project* | *Console*
- In the dialog select:
- *Simple executable (gcc)*
- Click *Next*.
- Give the project a name.
- Click *Next*. Click *Finish*.

Now add some code to the **main.c** file under the **src** directory shown in the Workspace panel.

Compiling a selected project from the Workspace

Question: I load the second project in one of the sample workspaces in *CodeLite* but when I compile it the first project is the one that is run.

Answer: *CodeLite* lets you 'activate' any of the projects in your workspace and it is the active project that is run, even when a file from some other project happens to be loaded into the editor. The way in which you activate a project varies according to which version of *CodeLite* you are using. In some versions, there is an icon at the top of the Workspace pane which, when clicked, lets you select the active project from a list. Alternatively, you can double-click a project name in the Workspace pane or you can right-click a project in the Workspace pane and select '*Set As Active*' from the popup menu.

Project settings and arguments

Question: In Step 2 '*HelloWorldArgs*' I can't find the project settings to add program arguments.

Answer: The way to load project settings has changed in recent versions of CodeLite. In some versions, you click a '*Project Settings*' icon over the Workspace window. If you can't see this icon, just right-click the name of the project in the Workspace window and select '*Settings*' from the bottom of the popup menu. Select *Common Settings*/*General* in the *Project Settings* dialog and enter arguments alongside '*Program Arguments*' in the '*Execution*' section.

No Output in Command Window

Question: I am running Codelite 7.0 on Windows. Every time I run the "Hello World" code all I get is a command window that states "*Press any key to continue.*" I press a key and the window closes without displaying "hello world". What should I do?

Answer: This normally occurs when the C compiler (*GCC*) has not been properly installed. This compiler is provided with the 'Minimalist GNU For Windows' (*MinGW*) development environment which may be installed with CodeLite, or may also be installed separately. Here are some things you can do to fix this problem:

- When installing CodeLite For Windows, select the **latest full installer** (*not* a beta or a weekly build!). Downloads are available here: http://downloads.codelite.org/
- If the problem persists, you may need to install GCC as a separate step. Instructions on doing that can be found here: http://codingfox.com/1-5-how-to-install-gcc-in-windows/

THE TERMINAL (OUTPUT) WINDOW VANISHES

Question: When I display output from my program, the command window vanishes before I can see what is in it.

Answer #1: Try using the CodeLite Terminal emulator instead. (Select *Settings, Global Editor Preferences, Terminal* and check *Use CodeLite terminal emulator*). Click *OK*. I'd recommend cleaning your existing build now (*Build, Clean Project*). Now rebuild and run (**Ctrl+F9**). You should see a terminal emulator window that displays any output from your program and then prompts to press a key to continue.

Answer #2: Also, bear in mind that in very short programs with no user interaction, the command window may shut before output can be viewed. Here's something you may want to try. Go to the final curly bracket in the code } - right-click in the left-hand margin and select *Add Breakpoint*. Now run under the debugger (*Debugger/Start Continue*) or press **F5**. The execution should stop at the breakpoint. The command window will appear (it may be behind the CodeLite editor so you may need to click its icon to bring it to the fore). This should now display any output in the command window. Go back into CodeLite and click **F5** to continue and end the program.

<PROGRAM> " IS NOT RECOGNIZED AS AN INTERNAL OR EXTERNAL COMMAND"

Question: I opened the **cmd** on Windows and entered the command:

03_HelloWorldArgs hello world

But I receive message *"03_HelloWorldArgs.exe is not recognized as an internal or external command, operable command or batch file."* Any idea what I can do to fix this?

Answer: This message means you are either in the wrong directory or you have not compiled the **exe** file.

You need to verify that the file that you are trying to run exists (take a look using Windows Explorer) and that you have opened the command window in the correct directory. For example, using Windows Explorer I want to verify that CodeLite has built my program in this directory on my PC:

C:\BitwiseCourses\COURSES\LearnC\SOURCE-CODE\CProjects\Step02\03_HelloWorldArgs\Debug

So, I log onto that directory and enter **cmd** into the Windows Explorer address bar. This opens a command prompt in the correct directory. I can verify that the **exe** file is there by entering:

dir

And sure enough I see: **03_HelloWorldArgs.exe**

To run it I now enter:

03_HelloWorldArgs

If you still see an error, you may not have correctly installed the C compiler). See 'No Output in Command Window' for more assistance.

XCode/iOS license error message

Question: In Codelite's 'Build' tab I'm getting this message, but I don't know what to do about it: *"Agreeing to the XCode/iOS license requires admin privileges, please re-run as root via sudo."*

Answer: You need to complete the installation of Apple's Xcode IDE. You must open Xcode once it is installed and accept the user agreement.

CodeLite doesn't have a C workspace option.

Question: Codelite 9.0 doesn't offer me the option of starting a new project in C, but only C++. Can I use this for C?

Answer: Yes. When you start a new *workspace* (which is a group that can contain one or more projects) you should choose C++. This lets you add both C++ and C projects to the workspace. When you add a new *project*, select *simple executable GCC*.

Using Other IDEs and Tools

NetBeans

Importing C projects into NetBeans

As I mentioned earlier, there are many editors and IDEs that can be used to edit and compile C projects. One important and widely used IDE is NetBeans. While NetBeans is probably best known as a Java programming IDE, it can also be used for other languages including C and C++. In order to use NetBeans with C, you must be sure to install a version of NetBeans with the C features enabled. You can either install the C/C++ edition or the full ('*All*') release of NetBeans; the *All* release would be my recommendation: https://netbeans.org/downloads/

Alternatively, if you already have a Java version of NetBeans installed and you want to add C support, you can do so by selecting the *Tools* menu, then *Plugins*. In the *Plugins* dialog, switch to the *Available Plugins* tab, scroll down to the *C/C++* item, check it and click *Install*.

Importing projects using a Makefile

The option to create a *Project with Existing Sources* may, at first sight, seem like the most obvious way of importing existing code files. I'll provide a quick overview of this before going on to explain my preferred method of importing existing code.

> Bear in mind that I do *not* recommend this method for importing small projects such as the ones supplied with this course. To import those, refer to the instructions in the section headed Importing source code into an empty project.

In principle, you can import an entire project or workspace by importing either a CodeLite *project makefile* (ending with the extension **.mk**) or the *workspace makefile* (just called **MAKEFILE**). To do this you would start a new NetBeans project, by selecting *File, New Project*; then choose *C/C++ Project* in left pane and *C/C++ Project with Existing Sources* in right pane. Then you could browse to the existing project directory and either select the *Automatic makefile* option to use the workspace makefile or the *custom* option to select a project (**.mk**) makefile. Be warned, though, this way of importing projects can be more complicated than it sounds. At the very least you are advised to build the projects in their current location using CodeLite

prior to attempting to import them into NetBeans. That is because the makefiles contain hard-coded path information. Alternatively if you are already familiar with using makefiles in NetBeans you could edit them by hand. But even if you succeed in importing the projects you may have problems building them due to various path and configuration options. In short, unless you already have experience of importing projects in this way and have a good understanding of using make files, I do not recommend that you use this method. However, if you decide to give it a try you can find more help here:

https://cnd.netbeans.org/docs/howto-existent/howto-exist.html
https://netbeans.org/kb/docs/cnd/quickstart.html#existingsourcesprojects

IMPORTING SOURCE CODE INTO AN EMPTY PROJECT

> If you need to import my sample code into NetBeans, *this* is my recommended method.

To import code using this method, you just create a new empty project and add the existing source code files to it. Here are the essential steps:

- Select *File, New Project*.
- Select the *C/C++* category in the left pane of the New Project dialog.
- Select *C/C++ Application* in the right pane.
- Click *Next*.
- Enter a project name such as (for example) *CDDatabase*.
- Browse to a location for the project (for example, *C:\Test*).
- Make sure *Create Main File* is **NOT** checked!
- Click *Finish*.

This will create a new NetBeans project in a new directory under the selected project directory. In the case above, this directory will be *C:\Test\CDDatabase*. The project contains no source code files. The next step is to add our existing code files.

Using a file manager, copy the code files from the original project (for example, the **.c** and **.h** files in the *Step10\02_CDdatabase* folder of the source code archive) into your newly created directory (*C:\Test\CDDatabase*).

In NetBeans, right click the *CDDatabase* project in the *Projects* pane and select *Add Existing Item*.

Select all the **.c** and **.h** files and click *Select*.

The files will now be added to the project.

Optionally you can arrange these files by dragging the **.c** files beneath the *Source Files* folder and the **.h** files beneath the *Header Files* folder.

Right-click the project node in the *Projects* pane and select *Properties*.

In the Properties dialog, highlight *Run* and change the *Console Type* property to *External Terminal*.

Click *OK*.

Now *Run* the project. The project should be built and will run in a popup Terminal window.

> PLEASE NOTE: I can provide no dedicated support for NetBeans or other IDEs. If you need help with using a specific IDE, refer to that IDE's documentation, its Wiki or its forum.

MICROSOFT VISUAL STUDIO

Windows programmers may use Microsoft Visual Studio to create C projects. In order to do so, you must be sure to install support for C++ projects. You may either use a commercial or a free edition of Visual Studio.

Visual Studio Community is the free edition. It includes support for programming languages including C, C++, C#, HTML/JavaScript, and Visual Basic. It is free for students, open source development, individual developers (creating *free* or *paid-for* applications) and small teams. Refer to the Terms & Conditions on the Microsoft site. Download here:

https://www.visualstudio.com/products/visual-studio-community-vs

To create a C project, follow these steps:

- Select the *File* menu, then *New* | *Project*
- In the New Project window select *Visual C++* (on the left) and (on the right) select *Win32 Console Application*.
- Name the project, and browse to a location on disk
- Click *OK*
- Click *Next*
- In the Application Settings dialog, check off *'Empty project'*.
- Click *Finish*.

Visual Studio now creates a new project with separate folders for Header Files, Resource Files and Source Files.

- Right-click the *Source Files* folder.
- From the popup menu, select *Add* | *New Item*
- Select *Code* | *C++ (.cpp) file*.

You will need to edit the default file name.
In the *Name* field, change the file name to: **main.c**
Click *Add*.

Edit the contents of this source file to the following:

```
#include <stdio.h>

int main()
{
	printf("Hello world\n");
	return 0;
}
```

To prevent the console from vanishing before you've had time to see the output, add a breakpoint by clicking in the grey margin to the left of this code line:

```
return 0;
```

A red dot should appear in the margin. Save the file and run (and debug) the application by pressing F5.

You can also create new projects from existing code by adding code files. Right-click the Source Files folder in the Solution Explorer and select Add | Existing Item. Naturally, header files should be added to the Header folder.

Compiling C From the Terminal or System Prompt

Question: I'm having problems using an IDE (such as CodeLite or NetBeans). Can't I just use a simple text editor and compile my programs myself?

Answer: Certainly you can. You could, for example, use a free programmer's editor such as Komodo Edit. Then open a command window in Windows or a Terminal on the Mac and compile your C code from the system prompt. You must open the prompt in the same directory or 'folder' as the C source code files. Then run the compiler (I'm assuming you are using the one named **GCC**) and specify the output (executable) file name after the `-o` option followed by the names of the C source code files you want to compile, like this:

```
gcc -o test main.c extracode.c
```

This command compiles the two source code files, *main.c* and *extracode.c* and creates an executable called *test* (or *test.exe*). The executable file can now be run by entering its name (in Windows) or entering its path followed by its name (on Linux and OS X) where `./` is the path to the current directory. So this is how I would run the executable file on Windows:

```
test
```

And this is what I'd enter to run it on OS X:

```
./test
```

See the lecture: '*Compiling C programs at the system prompt*' in Step 1 for more guidance.

> **Tip**: For simple ways of opening a command prompt in a specific folder on a Windows PC or on a Mac, be sure to see the lecture '*How to open a command prompt on Windows or OS X*' in step 2 of the course.

C Programming Questions

PRINTF – UNEXPECTED RESULT

Question: If I load the project *01_TestAge* and change this line…

```
printf("Your age is %d, so your bonus is %d.\n", age, bonus);
```

to…

```
printf("Your age is %d, so your bonus is %f.\n", age, bonus);
```

I get a very strange result:

```
Enter your age : 12

Your age is 12, so your bonus is
19881081741315001000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000.000000.

Press any key to continue.
```

What is going on?

Answer: An integer is 32 bit value, but a floating point (which you are attempting to display with `%f`) is 64-bit. Put simply this causes the program to read more data than is there (in your `int` variable) so it's reading in some unpredictable value. Remember that, unlike many other languages, C doesn't do much to prevent you from mistakes like this so you must be very careful to ensure that the format specifier in a string exactly matches the data that you are trying to display otherwise the results will be unpredictable.

CAN %D BE USED INSTEAD OF %F IN PRINTF?

Question: So since both `double` and `float` are floating points can't you use **%d** and **%f** interchangeably in a format string? But I noticed I got 0 for the numbers in the output when I put **%d** instead of **%f**.

Answer: %d is for integers (**d**ecimal integers) *not* for double. **%lf** (**l**ong **f**loat) can be used for *double*. The actual ranges and options available in format strings is quite complicated. This site goes into some of the nitty-gritty details:

http://www.codingunit.com/printf-format-specifiers-format-conversions-and-formatted-output

SCANF

Question: The `scanf` function seems like a useful way of reading in data. So why do I use `gets` in my code but not `scanf`?

Answer: `scanf` seems like the obvious partner function to `printf`. You can use `printf` to display specific a number of data elements each of which must have a specific data type. And you can use `scanf` to read in specific a number of data elements each of which must have a specific data type. However, there is a big difference. When you use `printf` in your programs, the number and type of data elements to be displayed are largely under you control. You can check that they are all valid and correct before displaying them. But when you read data (either from the console or from a file) you are relying on *someone else* (the user entering data or the person who created the file) to provide correct data. If you use `scanf` to read in the wrong data by mistake it can potentially cause catastrophic program crashes so it is generally regarded as a hazardous function. It is safer to read in ordinary string data with `gets` or a similar function and then parse out the data items in your code. This obviously requires more work than using `scanf` but it is also safer. More info here: http://c-faq.com/stdio/scanfprobs.html (If you want more examples, be sure to follow the links to longer explanations at the bottom of this page). There is also a discussion of the problems of `scanf` and a proposed alternative in this PDF document from the Physics Department of Ohio State University: https://www.physics.ohio-state.edu/~ntg/780/handouts/interactive_input_in_C.pdf

ARGUMENTS

Question: I noticed in one of the earlier lectures used default input arguments to **main()**:

```
int main(int arg, char **argv) { . . . etc.
```

So, given the above, why is there no error when *no arguments* are provided when running the program?

Answer: The C runtime initializes the arguments to `main()` if none are provided. In that case, `arg` is set to 0 and `argv` to `null`. So there is no error.

TRUNCATED DIVISION

Question: In the *01_Functions* program in *Step 5*, when 100 is divided by 3 the result shown is 33.000000. Shouldn't it be 33.333333?

Answer: In C when integers are divided the result is truncated (the remainder is discarded). You can force a higher level of precision (that is, to keep the remainder) by specifically casting the result to `double`. The code that does the division here is found in the `divide()` function. Rewrite this by adding `(double)` before the returned value, like this:

```
double divide( int num1, int num2 ) {
    return (double)num1 / num2;
}
```

UNEXPECTED RESULTS TRUNCATING FLOATING POINTS

Question: When I run the *01_Calc* program (*Step 3*), the value of the `taxrate` variable is shown to be 35 instead of 34 (which is the value shown in the video and in the section *Integers and Floats*, Chapter 3 of *The Little Book Of C*). Why is that?

Answer: It turns out that the result depends on several factors. Before I explain this, take a look at the code of this project.

```
#include <stdio.h>

int main(int argc, char **argv) {
    int subtotal;
    int tax;
    int grandtotal;
    double taxrate;

    taxrate = 0.175;
    subtotal = 200;
    tax = subtotal * taxrate;
    grandtotal = subtotal + tax;

    printf( "The tax on %d is %d, so the grand total is %d.\n",
                                  subtotal, tax, grandtotal );
    return 0;
}
```

As you can see, this does calculations with a mix of `int` variables and one `double` (floating point) variable, `taxrate`. The calculated values are all integers. When the value of `tax` is displayed, it may be either 34 or 35, depending on what

compiler you use, the optimizations used, whether the program is a 32-bit program or a 64-bit program and what processor you are running on. On CodeLite on Windows, the result is 34. On Visual Studio on Windows it is 35. On CodeLite on a Mac it is 35 too (though even these values may differ according to factors such as compiler optimizations).

The problem is that there is a loss of precision when the `tax` is calculated and the program may do different things to convert the floating point value of `subtotal * taxrate` to the integer value `tax`.

If you find that the result for `tax` is 35, try rewriting the assignment to `taxrate` by adding an `f` at the end of the floating-point number like this:

```
taxrate = 0.175f;
```

This has the effect of setting this value to a single precision floating point number (it is 'double precision' if there is no qualifier). Now, when I run the code in Visual Studio, the value is 34, not 35 as before. However, now I try this:

```
taxrate = 0.17500001f;
```

This time the answer is 35, even though it is single precision and it is near the limit of single precision accuracy (6 to 9 digits). That's what happens with the Microsoft C compiler but a different compiler may produce different results.

Looking into what's going on using the debugger (in either Visual Studio or CodeLite), I see that even though I make this assignment:

```
taxrate = 0.175;
```

…the actual value of the `taxrate` variable is 0.17499999999999999

Whereas, for this assignment:

```
taxrate = 0.175f;
```

…the value is 0.17499999701976776

So you can see that forcing a variable to be single precision even though the value assigned to it is double precision will have an effect. If you find it difficult to understand the details of the error here, don't worry – the problem with this specific bit of code is not important. The really important thing to learn from this example is, as stated in *The Little Book Of C*, that "An integer variable can only represent

numbers with no fractional part" so doing simple calculations and 'conversions' of floating point numbers to integers can very easily result in errors. The example above shows just how unpredictable those errors may be!

Implicit Declaration

Question: When I run the project *01_TestAge* on Windows, I get a warning: `implicit declaration of function 'atoi' [-Winplicit-function-declaration]`. What is that?

Answer: This may happen if **stdio.h** is not *explicitly* included. It will actually be included by the compiler anyway (which is why this is a warning not an error). You should be able to fix the warning by making the inclusion specific. That is, add…

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

…above the `main()` function and rebuild.

CHAR S[]

Question: Why are we allowed to use `char s[]` in functions, but when trying to use `char s[]` by itself the compiler complains - `array size missing in 's'`? When it's in a function is it an auto sizing array?

Answer: If you pass an 'empty array' as an argument, you are actually passing the *address* of the array (the location of the bit of memory where that array begins) rather than an actual array of characters. e.g.

```
void xxx(char s[]) {
}
```

But if you declare an array variable you need to set aside enough memory to hold a specific number of characters, otherwise it is an error to add characters as a specific index. So this is an error:

```
char x[];
x[0] = 1;
```

But this is ok:

```
char x[10];
x[0] = 1;
```

If you've come to C from another programming language, this may seem baffling at first as this is not how many other languages treat arrays. This is explained in depth in *Step 7* of the course and *Chapter 7* of the course book, **The Little Book Of C**. You just need to bear in mind that in C *an array is a memory location*. When you add items to an array you are adding items at adjacent memory locations. When passing an array you are passing its location (i.e. its address).

WHAT IS 'DEREFERENCING'?

Question: Some C programmers talk about 'dereferencing' and 'indirection'. What do these terms mean?

Answer: *Dereferencing* is just a fancy way of saying getting at the value that is pointed to. In C `*` is called the 'dereferencing' or 'indirection' operator. Try this:

```
int *ptr;
int x;
x = 50;
ptr = &x;
*ptr = 100;
x = *ptr;
printf("x=%d\n", x);
```

This uses indirection to 'dereference' `ptr` and put 100 at the location to which it points:

```
*ptr = 100;
```

It then gets that value by dereferencing the pointer again:

```
x = *ptr;
```

The only real problem here is that you must be sure that the pointer points to something before you dereference it. Trying to dereference a *null* pointer will probably crash your program.

In fact, you are also dereferencing when you access data at a memory location using square-bracket array syntax:

```
intarray[3] = 100;
```

In fact, you could even move to the location indicated by `intarray[3]` by simply adding 3 to the value (the address) of `intarray` like this:

```
*(intarray +3) = 100;
```

To see this, try this code:

```
int intarray[] = {100,200,300,400,500};
printf("1) %d\n", intarray[3]);
printf("2) %d\n", *(intarray+3));
```

You are also dereferencing when you use the `->` operator to access data in a `struct`. You can find examples of this in my *02_CDdatabase* program from Section 10 of this course (see also Chapter 10 of the course eBook, *The Little Book Of C*).

How do you print a constant long number?

Question: Using `printf()` I get a truncated number with the following examples:

```
#define LARGENUMBER 1.000000000000000079999

const long VERYLARGENUMBER = 2.000000000000000079999;
```

Answer: These aren't *large numbers*, but floating point numbers with *large precision*. The number of digits of precision you can get with the standard 64-bit C 'double precision floating point type (that is, `double`) is about 16 digits. In your examples, you have 23 digits which won't be stored by the machine - it will truncate them to 16 and you will lose the last few digits which is what you are seeing when you try to print them. Larger precision numbers can be handled, but you would either need special hardware or special libraries of C code.

malloc.h not found

Question: When I build a program on OS X that has the statement `#include <malloc.h>` I get an error message telling me that the file *malloc.h* cannot be found.

Answer: The C compiler and libraries supplied with Apple's Mac IDE, Xcode, put code normally found in *malloc.h* somewhere else. Comment out `#include <malloc.h>` and your program should run without error. If ever you have this sort of problem with other included header files you may need to check if there are any similar ways in which Apple reorganises the standard library source and header files. There is no absolute requirement that all functions and declarations are in particular files - but by convention they usually are. More information on this and similar Mac-related problems are documented here: http://macosx.forked.net/comp.html