Debugging and troubleshooting in depth

Before we start off on the adventuresome world of debugging and troubleshooting we need to get a few things straight. What is the difference between the two terms is something that you might want to know a little about first. Let's start off with what is debugging, well debugging is the act of finding and/or fixing bugs in code. So what does troubleshooting mean for us programmers? Troubleshooting is finding and fixing problems in software that come up during use. Excuse me, isn't that basically the same thing, yes and no. Debugging is strictly used to find problems in the code of a particular program. But troubleshooting is mostly done by users of programs (ever had a problem with browser, operating system etc.). Keep in mind that programmers can also troubleshoot, and many troubleshooting tips can be applied to debugging (maybe they don't apply exactly but what does at the end of the day). One good thing to remember about troubleshooting is that many problems are created by user error. Designing your application to be user friendly is a way a programmer can troubleshoot his program.

Let's go over a few tips and tricks in debugging before starting into the gory details……..

A few thoughts and basic tips on debugging and troubleshooting code:

The difference between a novice programmer and an experienced programmer is the ability to debug programs. It may or may not surprise you that even experienced programmers have bugs in their code. But an experienced programmer can quickly find and fix problems by way of debugging.

The following are a few tips and tricks……

Good Practice Tips

1. **Debugging structured code is much easier and more effective.** What does that mean you may ask yourself? Structured code means that you divide code into sub-units. Usually this is by means of functions/methods or classes. Then each unit can be tested to eliminate your bugs or problems. The structured approach also has the added benefit of allowing you to reuse code.

2. **Boil your code down.** Wait a second, how does that work. Well think of it this way, finding a needle in a giant haystack is going to be hard. But finding a needle in a tiny haystack is going to be much easier. So finding a bug in a huge program is hard, finding a bug in one function is easy. Eliminate code and try to narrow your problem down to a specific area.
3. **Make sure you question any assumptions that you make.** Unless you are perfect, which I doubt that you are, your assumption can be based off of imperfect observation. To put that another way lets define assumption a little better. An assumption is something that is assumed to be true with no defined proof, basically a guess. So don't get into the habit of assuming code works or isn't the source of a problem unless you can test and eliminate it.
4. **Be aware of the pitfalls of copy and paste coding.** Copying and pasting can save you time, but what if you introduce a bug in your copy and paste session. This bug may be repeated multiple times through your code. Which in turn may cause a myriad of problems that ripple through the cosmos and destroy the world. So try to keep in mind that this may not really save you any real time at all.
5. **Talk your problem out to an inanimate object or failing that a person.** Well no wonder people give programmers such strange looks. But seriously this form of problem solving is called rubber duck debugging (if you don't believe me look it up). This works by forcing you to explain your code line by line. In a word, forcing you to reexamine any and all code you have written. You would be surprised how this can help, so don't overlook its potential.
6. **The compiler lies, it doesn't know what it's talking about.** You may well have thought this a time or two, but let's think this one through a little. You fix the obvious compiler errors and only warnings are left. You decide you know best and ignore them. A few months later a nasty bug pops in for a quick chat. Ok, I will endeavor to explain a little better than that. Warnings may seem harmless, but they are often an indicator of a future problem. Hence the reason they are warnings and not errors. The compiler is not perfect but it does try. It is not psychic (somebody should work on that) but it does try to let you know when you are dropping a hammer on your foot. The moral of the story is try to catch bugs before they become a problem (watch out for the warnings).

A few final thoughts to chew on before you move on to bigger and better coding futures. Clean code and unit testing are a few words you may have seen in the programming world. You may know all about them but even so it is worth a mention (especially if you don't practice it). Clean code is at its heart readable. Doesn't seem all that special but keep in mind any code you write, you or another programmer will need to read and understand it later. So as an example let's consider the following variable names, **int Anynumber**, **int Zipcode**. Now the first example could really be anything, but the second example tells you exactly what it is. Another programmer (or even you given enough time) will not understand non descriptive variables and function names. So save yourself some hassle and use names that have meaning (other than to you that is).

Unit testing ensures that the smallest testable piece of an application (unit) is tested to ensure that it is fit for operation. Basically running tests on each little part of your program to see if it will work. This helps to find most errors, but not all. The hardest part to grasp sometimes is that tests are written before you write code. If you follow both of these practices you may never have to debug any code at all. It will just work the first time through which is sort of the point of programming.

Tips and Tricks of the trade

Herein you will find tips and tricks on debugging your code. Keep in mind these tips may not work on every language under the sun in all of the world. They are just a few good things to try if you need a little help. Therein, they may not be good practice so direct all hate mail to me at ………. uh never mind that last part.

1. **Wait, where did my application get that value.** Test values by outputting them so you can see what's going on in your application. Seems a little silly but you would be surprised how helpful this can be. So if a value seems off try throwing it into a printf, cout, label, messagebox and see what is going on in there.
2. **I like to program on the fly, so what if hello world takes me six months.** Try to plan your program out ahead of time, this helps to organize your code and your thoughts. Just setting things to paper can avoid quite a few problems that arise from coding on the fly.
3. **I wrote this but even I don't know what it is supposed to do.** If your code seems unnecessarily complicated, it probably is. Examine code blocks and cut down on unneeded code. Make your algorithms easier to follow and they will be easier to use down the road
4. **Programming Déjà Vu = that function seems real familiar, did I write this already.** Design code to be reusable and easy to integrate. If you have written something before than there is no need to reinvent the wheel. Just be sure that you understand its use and that it is well written to start out with.

   **Special note- If some code from the internet finds its way into your application (I'm sure no one has ever done that before). Please make an effort to understand it first before dumping it into your program.*
5. **When breaking your program is a good thing.** A breakpoint can help track down many nasty problems if you use them. They help to show where data is going and coming from, so keep them in mind when you debug. They are not always necessary but can be invaluable.
6. **A comment about a comment for a comment by a comment.** Even descriptive variables and functions only tell part of the story. A year or two down the line you may not remember what you worth thinking at the time. This is where a well-placed comment helps to refresh your memory and put you back on track. Not all code you write needs comments but a line here and there can save you quite a bit of time.

Finish by reading the article on http://www.codeproject.com/Articles/79508/Mastering-Debugging-in-Visual-Studio-A-Beginn