Chapter 1 – Introduction

Why I Am Writing This:

Why I am I writing a set of tutorials on compilers and how to build them? Well, the idea goes back several years ago when Rapid-Q, one of the best free BASIC compilers for Windows, died very suddenly when RealBasic bought the source and employed the sole Rapid-Q developer. Understandably, the people who had been using Rapid-Q started looking for alternatives and even considered creating a new, better Rapid-Q clone from scratch. I did not participate in these first compiler projects, because I did not have any knowledge of how to build a compiler at that time. Secretly, however, I did want to build my own compiler, so for the next couple of years, I searched the Net and the local library for books/articles/tutorials on the subject of compiler construction. I found almost nothing. Although I located many technical lecture notes on the subject, I could not even hope to understand them. However, I did find an old outdated tutorial named "Let's Build A Compiler" by Mr. Jack Crenshaw. In 16 chapters, it described very easily how to build a simple compiler using Turbo Pascal and a very old version of DOS. This really helped me understand the basic concepts of compiler technology, but left me wondering how this could be applied to today's world. For example, how do I translate these concepts into a Windows environment and into one of today's most powerful languages, C++? Laboriously by trial and error, I discovered; I also had to learn C++, Windows API, and assembly language in the process of building a Windows compiler, which wasn't always that much fun.

This experience left me with an attitude of "This would have been so much easier if I had had access to the right tutorials." Since then, I have search the Net for literally hours looking for free materials that might help average programmers learn how to build compilers, yet I still turned up absolutely nothing useful. Sure I found bits here and there, but nothing comprehensive. The only option for somebody interested in learning about this topic was to buy a $50 to $200 book! I found and still find this totally unacceptable as the Net is supposed to be the 'Information Superhighway.' As a result, I decided to fill this gap by writing this tutorial.

Aim of This Tutorial:

The aim of this tutorial, simply stated, is to teach the average programmer how to build a Windows toy compiler step-by-step using C++. Before I go into detail on this statement, let me tell you what this tutorial does not aim to do. First, this tutorial does not aim to provide a comprehensive view of compiler technology such as descriptions of various types of compilers or the theories that make those compilers work. I have seen and read those types of articles and I know that they put you to sleep almost instantaneously. Secondly, it does not

aim to teach you how to use the so-called compiler compilers or any other compiler tools such as Yacc, Bison, Flex, etc. Instead of trying to teach you how to use a certain compiler tool, I will teach you how and why compilers work and how to build one by hand, which, in my opinion, will equip you to build a compiler much more efficiently because you actually learn the internal processes that compilers use. Thirdly, this tutorial does not aim to build an optimized compiler, meaning the programs it makes won't be the fastest or the most efficient. However, they will not have to be interpreted like Rapid-Q and some versions of Visual Basic, so the overall speed should be about the same or even better than that of Rapid-Q. Fourthly, the compiler we will be building will not conform to any one language. It will be a mixture of BASIC, C++, and my own inventions. However, once you finish this tutorial, you should be able to build any type of compiler you want to, so I don't see this as a problem. Finally, I do not aim to show you how to do everything; some things I leave as 'an exercise for the reader,' meaning you will have to do some exploring on your own. If I tried to pack everything into this one tutorial, you would still be reading this when you died!

Earlier I mentioned that I planned to teach you how to build a Windows compiler from scratch using C++. First, let me answer some questions that you might have. Why build only a Windows compiler? Why not build a Windows, iMac, and Linux compiler? Well, the answer is two-fold. Primarily I chose to limit the compiler to just Windows because I don't know how to program on any other OS. Also, most programmers come from a Windows background (I wonder if Visual Basic, Delphi, and Visual C++ have anything to do with this?).

So why use C++ when plenty of easier alternatives exist such as Visual Basic or Delphi? When I first began to build my compiler project, I didn't want to invest much money, so that eliminated languages such as Visual Basic, Delphi (Yes, I do know about the free Personal Edition of Delphi, but that didn't exist when I made my decision), and other commercial options. In addition, I wanted a fast language with the capability for large projects, which made choices like Rapid-Q seem unlikely. Finally, I wanted a powerful language that was cross-platform, meaning I could take the code and modify it to work on another Operating System such as Linux if I wanted to do so in the future. That narrowed my choice down to C++ and a few other languages, and when compared, C++ won out. Now I know you are probably saying, "I don't want to learn C++, that's a lot of hard work; besides, that language looks like a scrambled pile of semi-colons, curly quotes, parenthesis, and weird words like 'void' and 'virtual' and, heaven forbid, 'polymorphism'!" Well, I agree, C++ does have a bad reputation, but it gets the job done quickly and efficiently, which is a must for compilers (unless of course you like waiting 10 minutes for your latest project to compile!). Although C++ does take a bit of time getting used to, it isn't that hard to learn and we don't have to learn much of the language to build a compiler, so bear with me.

What I Am Assuming (About You):

Before we start this tutorial, I am assuming several things about you:

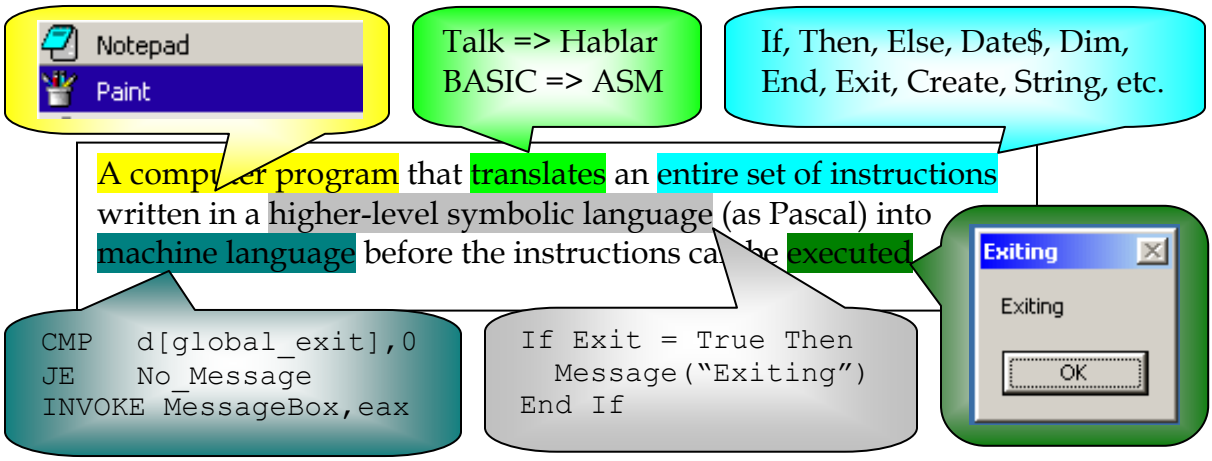
1. First, I am assuming that you know what computer programming is and have done a bit of programming in some language such as Visual Basic, Rapid-Q, Delphi, or maybe even C.
2. I am assuming that you have used a compiler such as Visual Basic, Delphi, Rapid-Q, or Dev-C++ and want to learn how they work by building one yourself.
3. Lastly, I am assuming that you are willing to work hard and try to learn the material presented here. I will try to make it as easy and straightforward as possible, but some parts might not always be easy.

What a Compiler Is:

Before we get started building our compiler, I think a brief overview on what a compiler is would benefit us. So then, what exactly is a compiler? The Merriam Webster dictionary defines the word 'compiler' as "a computer program that translates an entire set of instructions written in a higher-level symbolic language (as Pascal) into machine language before the instructions can be executed." Whoa! That is quite a confusing statement (you wonder why Webster hasn't gone out of business yet with definitions like that!), but lets try breaking it down into little pieces. The first part is simple enough, a compiler is a "computer program" just like Microsoft Office, or Notepad, or any of the apps you might have programmed in Visual Basic or Rapid-Q. The next part says a compiler "translates." Let us see here, translates means it takes one thing (like the English word 'talk') and somehow turn it to another thing (like the Spanish word 'hablar'), but with both things meaning the same thing (a verb that describes the act of communicating with voices). Webster also says that a compiler translates "an entire set of instructions." That's a good thing! I should hope that a compiler could do more than understand only a single instruction like 'shut down the computer'! Now the definition gets a bit tricky: the compiler has to translate "instructions written in higher-level symbolic language (as Pascal) into machine language." Yikes! That looks pretty complicated, but you just have to know what it means. "Higher-level symbolic" language just means a programming language such as BASIC, Pascal, or C++ that a normal person can look at and easily understand. Machine language isn't like that; it looks like pure garbage to us humans because only the computer has to understand it. That's where a compiler comes in handy. A compiler can look and the "higher-level symbolic language" command "shut down," and then turn it into something the computer can understand. That saves us humans from having to tell the computer what to do in its own language. The final part of the definition says that the instructions are then "executed", which means the program is run. To sum it up, a compiler takes

a whole language (BASIC for example) and turns it into another whole language (machine language for example) so you can make programs. As an analogy, a compiler performs the same job as a foreign language translator – taking one language (English for example) and turning it into another language that somebody else can understand (Chinese for example). The compiler just uses different languages; it will translate BASIC to assembly language (a type of machine language).

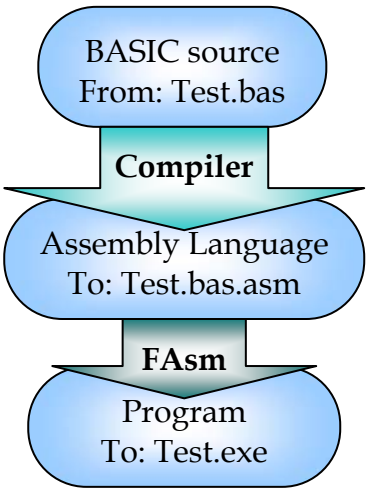
For those visual learners out there, here is a diagram for the definition:



How It Will Work:

This section shows you how a compiler, in general, works, and more specifically, how our compiler will work. As you learned in the last section, a compiler takes one language and turns it into another. That begs a question: what languages will our compiler translate from and to? Well, I thought about that and decided that our compiler should take a form of BASIC and turn it into machine language in the form of assembly language (Just like in "Let's Build A Compiler"). I am going to assume (Yes, I know that is usually a bad thing to do) that you know what BASIC is, but you might not know what assembly language is. Assembly language is a programming language just like BASIC, except it is really hard to learn. However, assembly language is super fast and compiles into really small programs, which makes it a good choice. I will show you the easy parts of assembly language, and

Visually, our compiler will do this:



when I do, I will thoroughly explain everything, so don't worry. In order to get from our BASIC source code (the actual text) to a Windows program, we will have to go through several steps.

First, as you can see on the diagram on the right, we will start off with BASIC source code. This is similar to a Delphi Project, a C++ *.cpp file, or a Rapid-Q *.bas file. This file contains the text that describes how our program will operate.

Secondly, just like Rapid-Q and any other proper compiler, our compiler will turn this BASIC source code into machine language (in the form of assembly language). This is the actual compiling process that we will concentrate most on learning.

Thirdly, we will use an assembler to take the assembly language and turn it into a Windows program. This saves a whole lot of work on our part (you can now sigh in relief). The assembler we will be using, Flat Assembler (FAsm for short), is not a standard assembler like the Microsoft assembler or the Netwide assembler. Its size (200 KB) makes it much smaller than the Microsoft assembler (over 20 MB!), which reduces our overall compiler package size quite a bit. That's the main reason why I chose it.

What You Will Learn:

So what exactly are you going to learn during the course of this tutorial? Well, for starters you will learn a type of BASIC, C++, Windows API, assembly language, and compiler technology. That's the easy part; then you learn…just kidding! That's all you'll be learning (and believe me, that's going to be enough). Here's a basic outline of how we will use these skills to build a compiler.

- C++ — We will use this language to program the compiler.
- A type of BASIC — We will create this language for our compiler (like William Yu created Rapid-Q language and like Microsoft created the Visual Basic language)
- Assembly language — Our compiler will turn the BASIC into machine language in the form of this language.
- Windows API — Our assembly language programs will need to use this to manage memory, create windows, and just about everything else.
- Compiler Tech. — Sort of need that to build a compiler!

Conclusion:

Well, that concludes this first chapter. Now that you have an idea of where we will be heading, what we are trying to accomplish, and how we plan to accomplish it, we will take a short break and pick up here in the next chapter. To give you an idea of where we will be heading, in the next chapter we will start planning the basic structure of our compiler. In addition, we will start to learn a

bit of C++ programming and begin to lay the framework for the compiler, but more of that in the next chapter. See you then!