

Polishing That Rusty CW

By
Jack Purdum, Ph.D.
W9NMT

Over fifty years ago I remember getting my Novice class license after passing a 5 word per minute (wpm) code test and a simple written exam. I spent the next year chasing contacts all over the world with my Heathkit DX-35 cranking out about 65 watts input into a simple dipole antenna. Of course, every Novice on the planet couldn't wait to pass their General Class exam so they could "get on phone". I remember Charlie (another Novice friend of mine, both of us age 13 at the time) and me taking the bus to Cleveland to take our General Class exam.

We were terrified.

Both of us felt fairly confident about the written part of the exam, but the code test had us worried. We had pushed our code speed up over 15 wpm, but we knew we were shaky at that speed. The rules were that you had to copy and send 13 wpm error free for at least one solid minute out of five. The receiving test came first. As I recall, about 30 of us were each given a pad and pencil and were seated in chairs that were hidden behind three long rows of dark tables in a large room painted Hospital Green. It was not a friendly place. After a few instructions by the examiner, the code test began.

Good grief! The room had the acoustics of a steel box. The first letter sang out from the speaker on the front table only to be bounced off the back wall and into your ear at the same moment the next letter came from the front of the room. I glanced at Charlie and his pie-plate eyes told me he, too, was having a mild coronary because of the echo. With a heartbeat that outpaced the code speed by a factor of 20, we tried our best to copy at least one minute of the echoing code. At the end of five minutes of code, we were told to put our pencils down, turn in our code sheets, and wait in the next room.

About two and a half days later (actually, I was told it was only 20 minutes), the examiner came out and said he would read off the list of people who could sit for the sending part of the code exam. Alas, Charlie didn't make it. I was lucky. My code sheet had a big red circle around what amounted to 65 letters of correct code that I had copied. The sending test was a snap and I got to sit for the theory part of the exam. Nine weeks later, I had my General Class ticket. (Charlie passed his exam on his next try.)

As of February, 2007, the FCC dropped the code requirement from all Amateur Radio licensing requirements. In a way, that's sad. William Packard, NN9U, recently wrote an article¹ giving some very good reasons why you should know, and use, Morse code. (Charlie and I played football together later in high school and used Morse code to relay blocking assignments to each other...a use not on Bill's list.) The purpose of this article is to help you make the move to

1. William E. Packard, NN9U, *Morse Code: Efficient Or Over the Hill?*, **QST**, January, 2009, pp.55-58.

(or back to) CW. You really should give CW consideration for the reasons mentioned in Bill's article plus it can be very enjoyable with a little practice.

Getting the Rust Off

I just retired in May of 2008. While I've been continuously licensed since 1954, for about the past 8 years I've only been active enough to maintain my ticket. Now, I want to get back to ham radio operating with a lot more vigor than I have in the recent past. My code speed is pretty bad because, quite honestly, I haven't done a lot of CW work lately. Still, I do want to get proficient at it again. NN9U's article just pushed me over the edge.

So...what's the best way to either learn CW or polish your old CW skills? Obviously, practice is the *only* way to learn or improve your CW skill set. Bill's article correctly points out that, if you want to boost your code speed, copying letter-by-letter isn't going to cut it. You can copy code that way to a point, but then you physically can't write down letters fast enough. Indeed, I think there's some unwritten law that says letter-by-letter code copy is inversely proportional to your age. You need to start listening to code as word patterns, as Bill explains in his article.

I've been listening to QSO's on the air for several months now, and my receiving code speed has improved somewhat. However, trying to improve my code speed this way is time consuming for several different reasons. First, most CW contacts that you eavesdrop on tend to have the same content (e.g., name, QTH, RST, etc.). There is often not enough variety of words to really practice your CW ear unless you get lucky and stumble onto a nice rag chew in progress. Second, you're at the mercy of the band. There are days when there are very few stations to be heard clearly on the CW portion of the bands. Finally, if you do find a station you can copy, it may or may not be sending at a speed that fits in with your desired code speed goal.

W1AW does have scheduled times for code practice (see <http://www.arrl.org/w1aw.html#w1awsked> for more details). You can also download MP3 files from the W1AW web site and practice listening to those. For me, this still wasn't a perfect fit because band conditions often screwed up listening to W1AW on the air, plus the times that I could devote to listening didn't always fit their practice schedule. The MP3 files get around these limitations nicely, but still has one limitation for a left-brained guy like me: I need to see the code while I hear it. Also, I prefer a lower pitch than provided in the MP3 files.

So, what's the solution? For me, it was to write a computer program that would send practice code for me. The design goals were pretty simple. The program must:

- 1) Allow variable code speeds
- 2) Allow the pitch of the code to be changed
- 3) Allow practice input to come from the keyboard or from a disk file
- 4) Allow me to see what's being sent as it's being sent

With those objectives in mind, I set about writing a program to fulfill those goals.

Using the Program

Figure 1 shows the user interface for the program. Near the top of the program screen is a spin control that lets you adjust the speed of the code. You can use the small arrows on the right side of the input box to increase or decrease the speed, or you can just type in whatever speed you want. Right next to the speed control is a textbox that allows you to change the pitch of the tone used to send the code. I have no clue what note the value 538 corresponds to, but it's pleasant to my ear. You can click on the *Test Pitch* button to hear what the value you've entered sounds like.

If you look closely at the words that appear in the large textbox, you will see that they correspond to the words from Table 6 in Bill's article. That is, they are the most common four-letter words used in the English language. As Bill suggests, you should get used to listening to such common words at the code speed you wish to use. However, it would get very tedious to retype all these words each time you wanted a practice session using these words. For that reason, I stored the words in a disk text file and use the *Open File* button (upper right corner in Figure 1) to read those words into the upper textbox. Anytime I want to practice these words, I simply load the program, set the speed and pitch, and click the *Open File* button and navigate to the practice text file I want to use. I've made copies of Tables 4 through 7 from Bill's article.

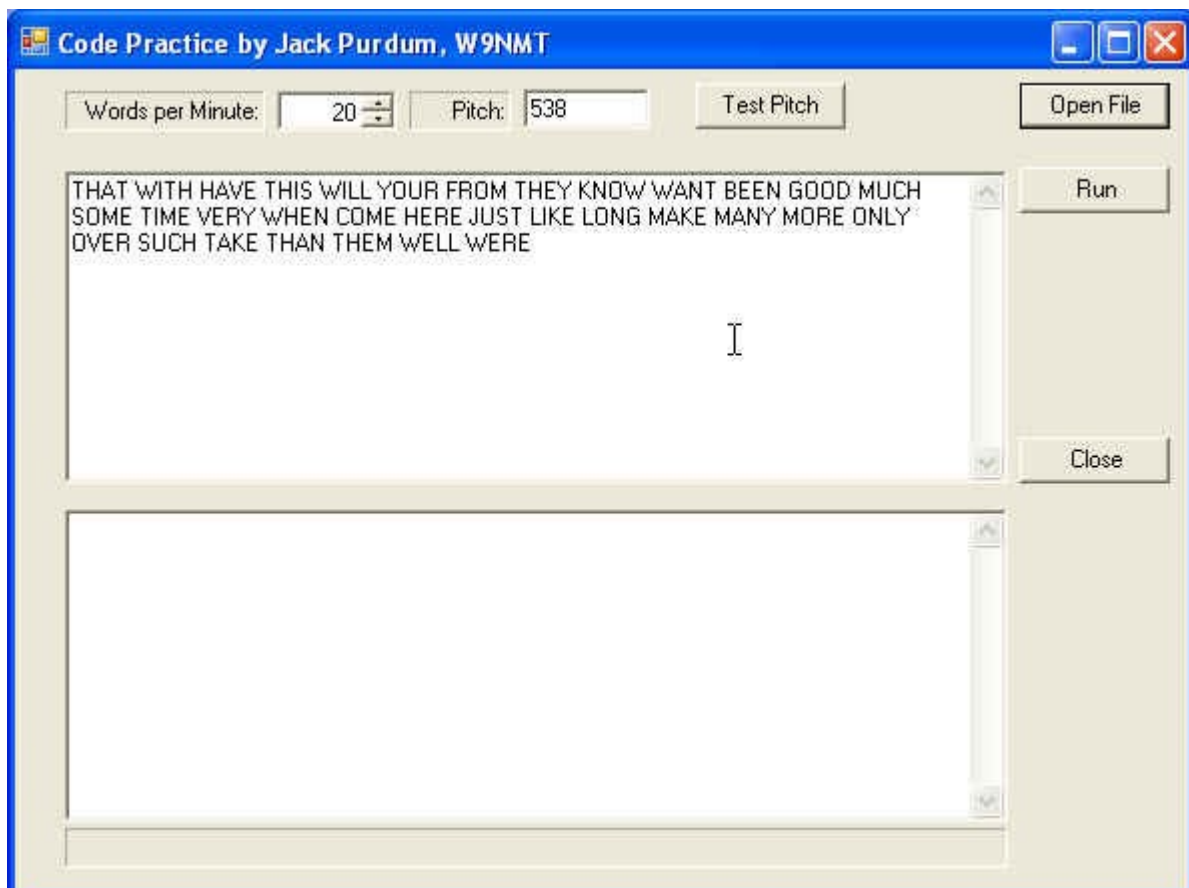


Figure 1. The User Interface for the Code Practice Program

The file can be any text file you might have. If you have an email, letter, or anything else written as a text file, you can use it as input into the program. Note: word processor programs, like Microsoft Word and others, have all kinds of embedded characters for print formatting that make direct use of such DOC files inappropriate for using as an input file. However, you can load a formatted document and use the SAVE AS feature to save the file as a “pure” text file. That text file can then be used in the code practice program. Another alternative is to cut and paste from a formatted document to the Clipboard and then copy from the Clipboard into the input textbox of the program, the program should work just fine. Any embedded formatting characters are stripped out during the copy process.

If you want, you can skip the Open File feature and simply type in whatever words you wish into the upper textbox and use that for practice. The program functions the same whether you use direct keyboard input into the textbox or read the contents of a text file into the textbox.

Once you have the inputs (i.e., speed, pitch, and text) set, simply click the Run button to start the practice session. The program outputs the tone through your computer’s speakers using the pitch and speed you selected. Figure 2 shows a sample run after all of the words have been sent. The output seen in the lower textbox in Figure 2 appears character-by-character as it is sent. If you press Run a second time, the lower output textbox is cleared and the process repeats itself.

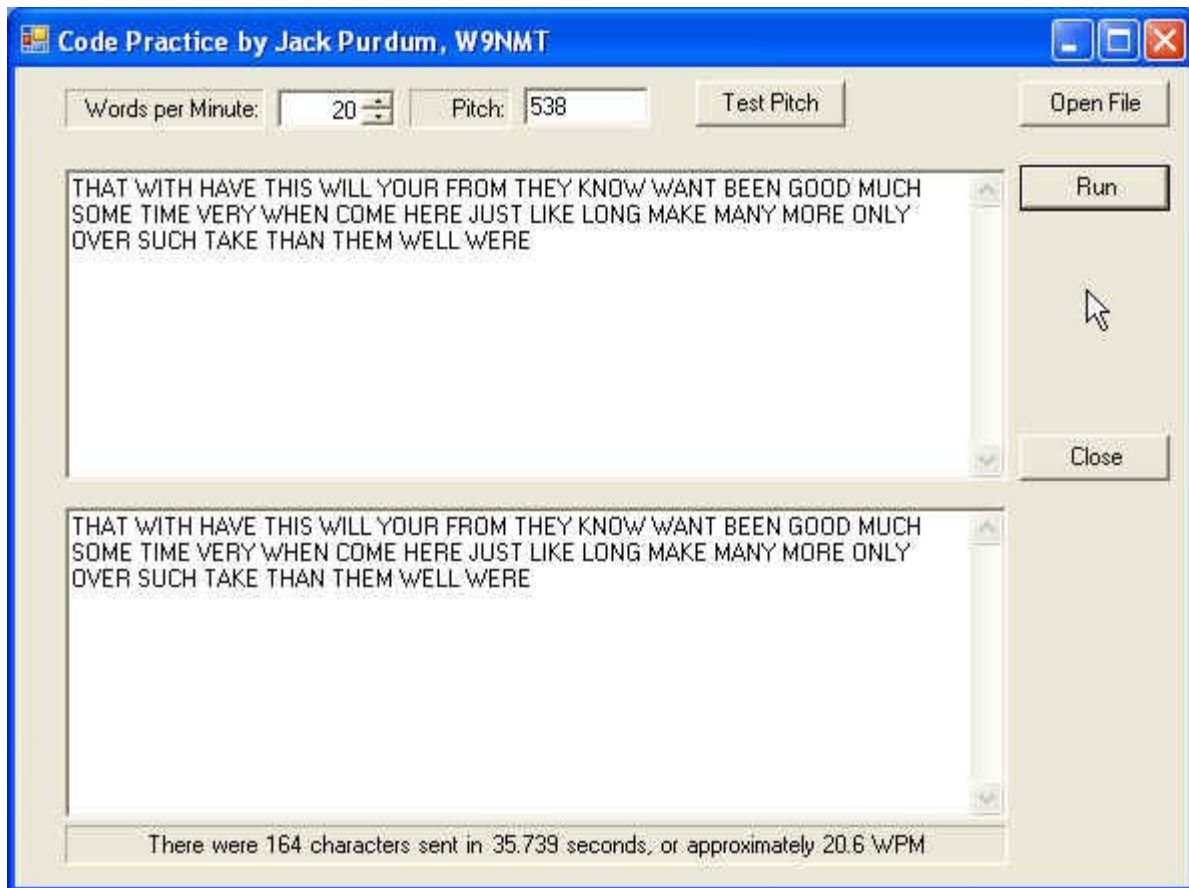


Figure 2. Program Output After Sending Several Words

At the bottom of Figure 2, you'll see some summary statistics about the text that was just sent. I added this output because of some technical hiccups I noticed while developing the software. More on that in a moment.

Some limitations

There are some limitations built into the program, but none of them are that serious. If there is an avalanche of complains about a restriction or some missing key feature, let me know and perhaps I can integrate it into a revision of the software.

There is a limit to the characters that the program can recognize in the input textbox. The program only recognizes: 1) alphabetic letters A-Z, both upper and lower case, 2) the digit characters 0 through 9, and 3) punctuation marks for period, comma, and question mark. It's my experience that these limits should cover about 99% of any character set you are likely to use during a QSO. If a character in the input textbox is not within the defined character set, it is displayed in the output textbox, but no audible tone is generated. If you try to enter lower case letters directly into the upper textbox, they are automatically coerced to upper case letters.

A second limitation is the keying speed range. The problem creeps in and becomes more noticeable at higher speeds, especially above 40 wpm. The overhead associated with the

processing of each letter (see limitation three below) becomes a larger component of the processing time for each letter of text. While I can't copy code at 40 wpm or higher...at least, not yet...it just sounds like it's not quite "right" at higher speeds. Still, it's probably good enough for practice.

A third limitation is the calculated keying speed. If you opt to practice at 20 wpm, the truth is that what you hear is likely not going to be exactly 20 wpm. The value, however, is within a fairly narrow error range. Because of this imprecision, I added the bottom line as seen in Figure 2. While I opted for 20 wpm during the run in Figure 2, I actually got about 21 wpm...probably close enough for government work. There are a number of reasons for the speed error.

One reason is because I used threading to write the program, which has some processing overhead associated with it. Second, despite your processor's speed, some time must be used in converting the text to an audible signal. In my algorithm, I use an array of strings to store the letters rather than an inline sequence of nested *if* statements to select the proper Morse character. For example, I store the letters 'A' and 'B' as "13" and "3111". This is because a "dash" is three times longer than a "dot". Coding the Morse characters this way allows me to use the string to figure out the proper tone length when sending each character.

Because the letter 'A' is stored in the first position in the Morse code string array, 'B' in the second and so on, I can use the ASCII (American Standard Code for Information Interchange) codes to index into the Morse code array. You'll note that all text in the upper textbox appears in upper case regardless of the case used to enter the text. Because the ASCII code for the letter 'A' is 65, if I read an 'A' from the textbox, subtracting 65 from its ASCII code gives me the appropriate index into the Morse code character array for the string representation for the letter. For example (I use C# to write the program):

```
characterToSend = 'A';  
  
arrayIndex = (int) characterToSend - 65;  
  
SendCharacter(morseCodeArray[arrayIndex]);
```

If you follow this code, you'll discover that *arrayIndex* is 0 which causes the letter 'A' to be sent. While this takes a while to explain, it executes fairly quickly...but not instantaneously. For this reason, there is a slight difference between the desired wpm and the actual wpm. I used the system timer to create the display message seen at the bottom of the screen.

The dot length is determined by the equation:

$$\text{dotTime} = 1200 / \text{wordsPerMinute}$$

where *dotTime* is in milliseconds. The Windows *Beep()* system call allows you to pass in the pitch and duration for a tone you wish to produce. For example, if you want to send code at 20 wpm, each dot should be 60 milliseconds in duration. Once this *dotTime* value is known, the other spacing is determined by CW convention. I have maintained the 1:3 dot-to-dash ratio, a single dot for element spacing within a character, a 3 dot spacing between characters.

Convention requires a 7 dot spacing between words, but I lowered this value to 5. Because of the overhead code associated with indexing into the array, fetching text from the textbox and the associated threading going on, each new letter has some overhead associated with it. Trial and error showed that lowering the word spacing value to 5 resulted in a pleasant-sounding “fist” and produced actual code speeds near the desired sending speed. Messing around with the inter-character spacing did not produce the desired results.

Conclusion

I’ve been using this program for just a few hours and already I am beginning to hear “words” rather than letters. For my ear, the word “been” always brings a smile to my face every time I hear it. Don’t know why...it just does, and it was the first pattern that I recognized easily. You can download the program for free from the ARRL web site at www.arrl.org/go?7
Give the program a try and soon you’ll be enjoying the fun that only CW can afford you.

About the author

Dr. Purdum is a Life Member of the ARRL and is recently retired from Purdue University’s College of Technology. Dr. Purdum is the author of 15 computer programming books and numerous related articles. He is looking forward to becoming more involved in ham radio activities in the future. He may be reached at jjpurdum@yahoo.com.