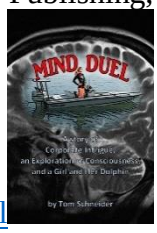
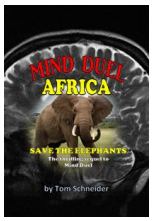
Programming & 3D printing
(for Everyone)

abstract

This is a great tutorial for anyone who wants to venture into the worlds of either 3D printing or computer programming, especially those who fear they don't have the technical expertise for such things. I write with the goal that any novice can understand the content as I present it. You experienced geeks might find something useful, too.

I also hope that in appreciation for my effort, you will treat yourself additionally by reading Mind Duel, my novel, which has nothing to do with the subject of this tutorial. Follow the link to my sponsor, Dual Minds Publishing, to order the book(s).





https://dualminds.org/mind-duel

Made using my techniques:


Programming & 3D printing
(for Everyone)

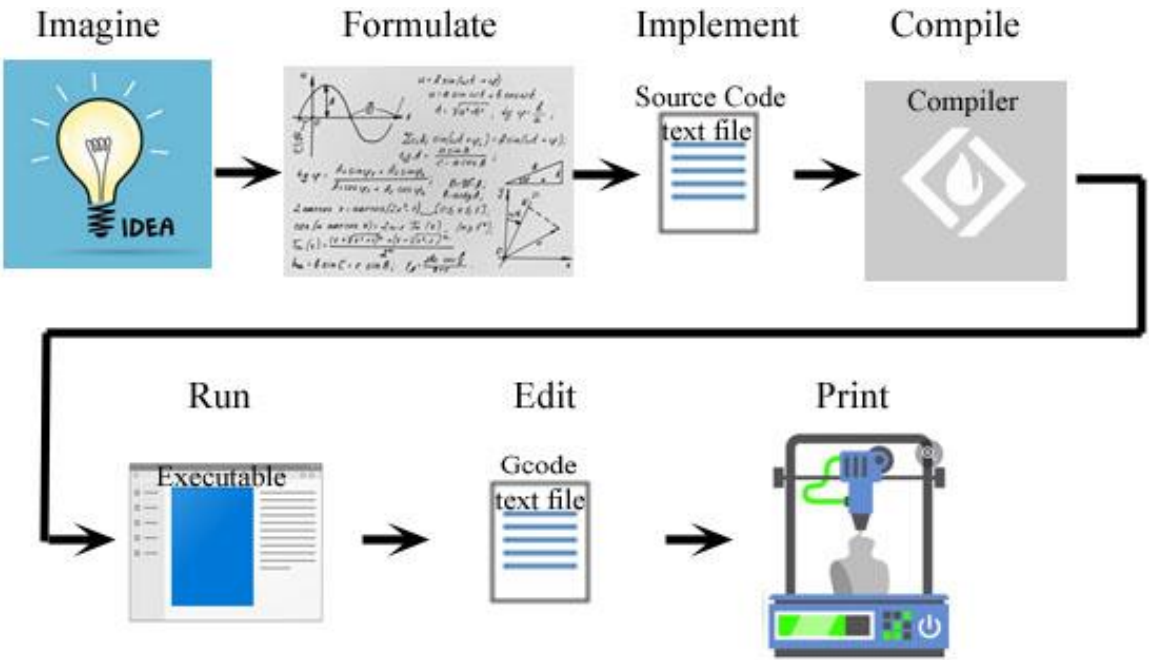
Overview:

Every time I search for help with a process involving computers, I find instructions that invariably leave out some little moronic step that, when left out, the process fails. It's as though the geeks providing the info just want to be smart-alecs, or they want to preserve this private expert's club that only they will have access to. They don't really want you to succeed. The goal of this guide is to cover every step of the programming and printing process, and it is written for anyone, irrespective of training, education, or even familiarity with computers, to be able to use computer programming and a 3D printer. Pretty amazing, huh?

More specifically, this document details every step required to make a 3D printed object, from an idea initiated from either a pencil sketch or a set of geometric equations, and implemented via Fortran programming. You can do it!

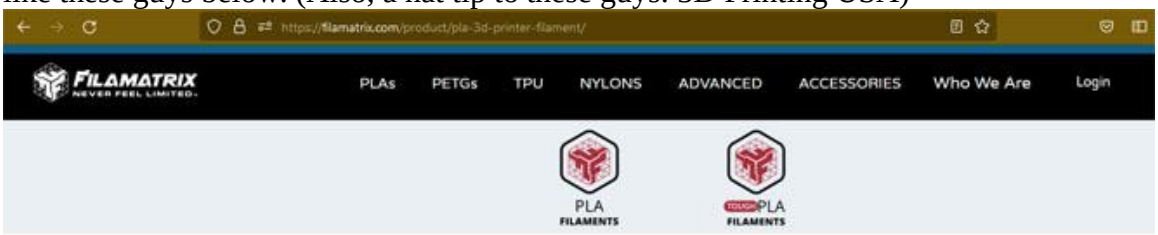
Note: There really isn't that much to learn here, but the writing may seem long and tedious because I will leave nothing out. I hate it when someone greatly underestimates my stupidity and proceeds with instruction with the assumption that I knew all the prerequisites, when, in fact, I'm woefully uninformed.

Your process is probably even simpler than this graphic overview of what you will do:



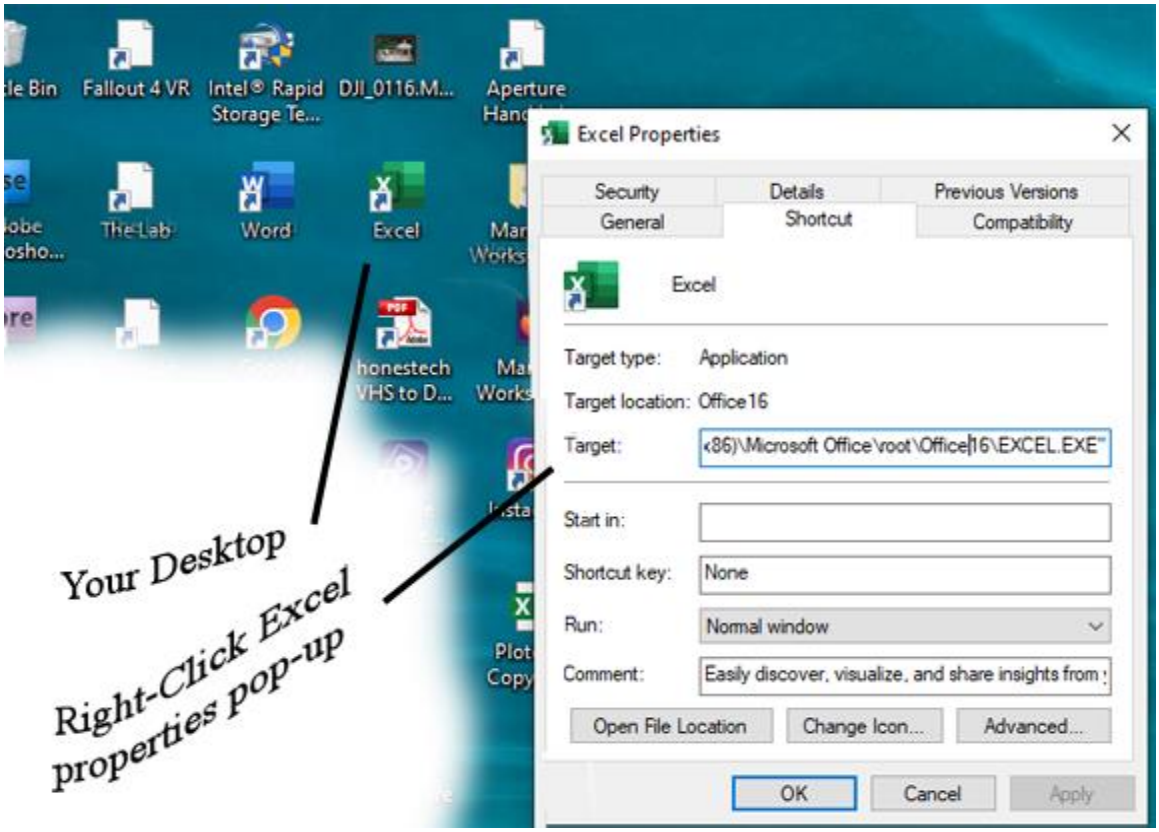
First, you need a 3D printer. This is the one my sons gave me as a gift that started all of this. (Don't think you can find this price anymore, though)

You will also need filament to feed your printer. It uses PLA, polylactic acid, and it is safe and environmentally friendly. You can find other colors and finishes from other sellers, but I like these guys below. (Also, a hat tip to these guys: 3D Printing USA)



Now you have your printer and filament. Most beginners now download an object file from the internet (see Thingiverse.com), but we're no beginners, right? We are going to dream up something ourselves. For this you will need to learn computer programming, the easy way. I

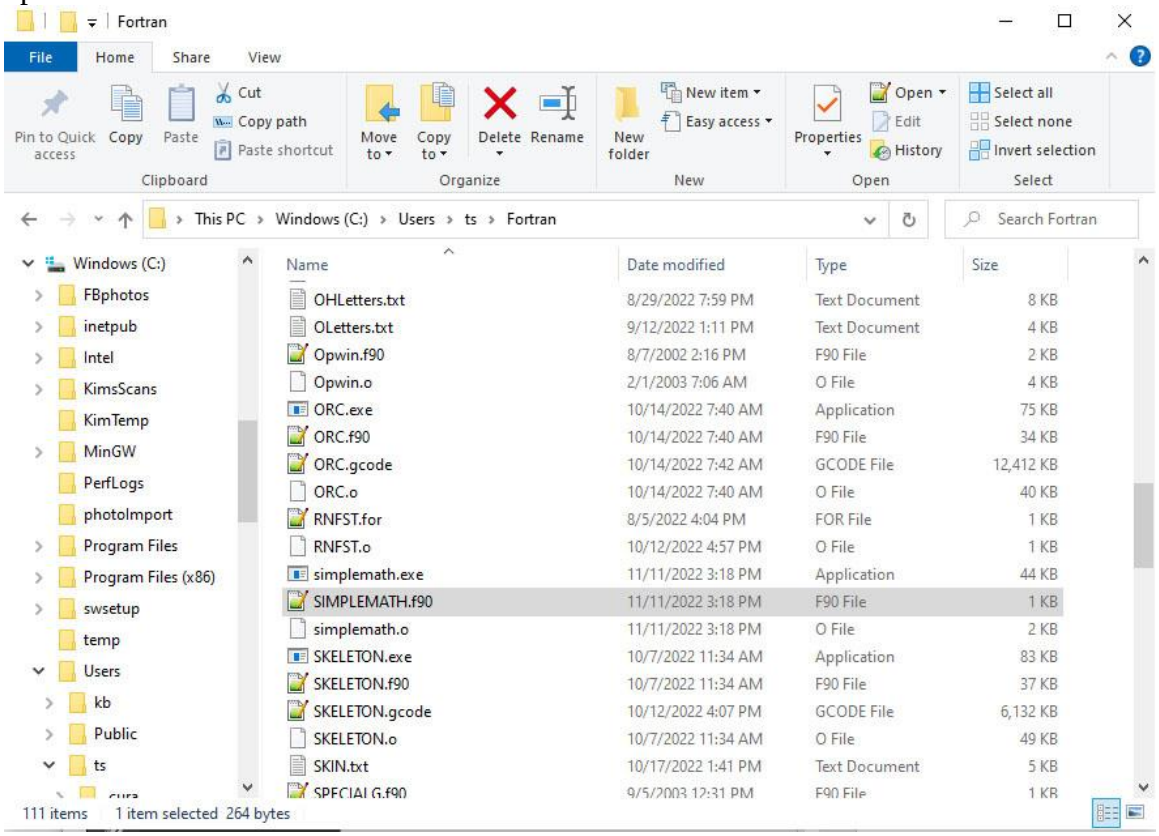
promised not to leave even the smallest step unexplained, so let's take a really big step back and review personal computer basics. Your desktop (the opening home screen on your computer monitor) has icons you can click with the mouse (how's that for basics!) Most of these icons are hot points on the screen such that a double-click on them will tell the computer to go launch a particular executable. For example, double-click the Excel icon and a thing labelled EXCEL.EXE will start running. You can see how this is set up by using the right-side button on the mouse to "right-click" the Excel icon and then selecting the "properties" choice in the pop-up menu that will appear. There you see the "Target:" is specified to be EXCEL.EXE to be found in the path,
"C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE"



We have broached two new subjects here that will be important: 1) paths, files, and folders, and 2) executables and documents. We can generalize that everything on your computer is a file, which to your computer is a discrete partition of memory storage consisting of 1's and 0's. Some of these files are special types called executables, typically with the ".exe" "extension" in the name. The extension is the part of a file name comprised of a dot and then a descriptive short text that informs of the type of file. This particular extension is for executables and tells the computer to go copy this file's 1's and 0's into RAM when the icon is double-clicked or the executable is otherwise invoked, and there start following the programming instructions effected within. RAM is that random access memory silicon chip where all the computing is done. This is distinct from your regular computer memory where your programs and data, executables and documents are stored. Thus, you launch the spreadsheet program called Excel by double-clicking the Excel icon. (If you have an internet-based version of Excel, you may not see this path.)

Because using the term “computer program” is somewhat ambiguous, we will define two terms to make an important distinction. The computer program source code will be a text file that you can open, read and edit. The computer program executable will be a file in the computer that actually does the work. The computer no longer needs the source code once the source code has been used to create the executable. Building an executable from source code is called compiling. The executable is the thing on your computer that actually runs, reading input and creating output. Most of those icons on your screen that you double-click to launch an application are linked to an executable in a file folder on your hard disk. Most “document” types (files that aren’t executables) are also linked to a particular program such that “opening” them will also launch the program that created it. That is to say, you can launch Excel by double clicking either the Excel icon or an Excel document.

One more little tedious topic to make sure no one is left behind, folders and paths. A file is physically located somewhere in the computer’s memory, and for our benefit, the computer kindly lets us organize where we keep it. We can specify a “folder” that is in a “path” to where the file is. Of course, many files can be kept within a folder, and folders can be kept inside of folders. Use Windows File Explorer to see and use the “folder tree structure”. If there is not an icon on the desktop for it, right click the Windows logo in the extreme lower left corner and select File Explorer from the popup menu. A window something like this will open.



There are quite a few computer programming languages. The best, in my 1970’s pre-totally geeked out computer world opinion, is Fortran. Easy and logical. No secret boy’s club. Warning: If you are online reading the literature and they are talking about “building” your

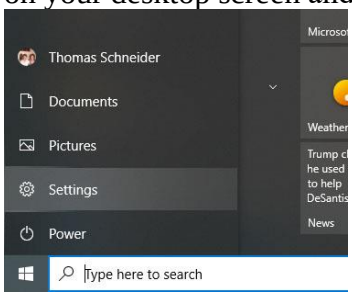
Fortran, you are off in the weeds. You do not want to “build” your Fortran development environment. You likely can’t, even if you want to, without a year or more in geek school. You just want to get hold of (download) the ready-working Fortran compiler application. A “compiler” takes your code written in a computer language in a simple text file and helps you create an executable.
And, here is how you get that ready-to-go compiler!

Go to SourceForge (By this I mean: 1.own a computer with internet access, 2.Double-click your favorite browser application e.g. Firefox or Chrome, 3. Type the following “uniform resource locator” (URL) into the header text field and press Enter.)

https://sourceforge.net/projects/mingw/

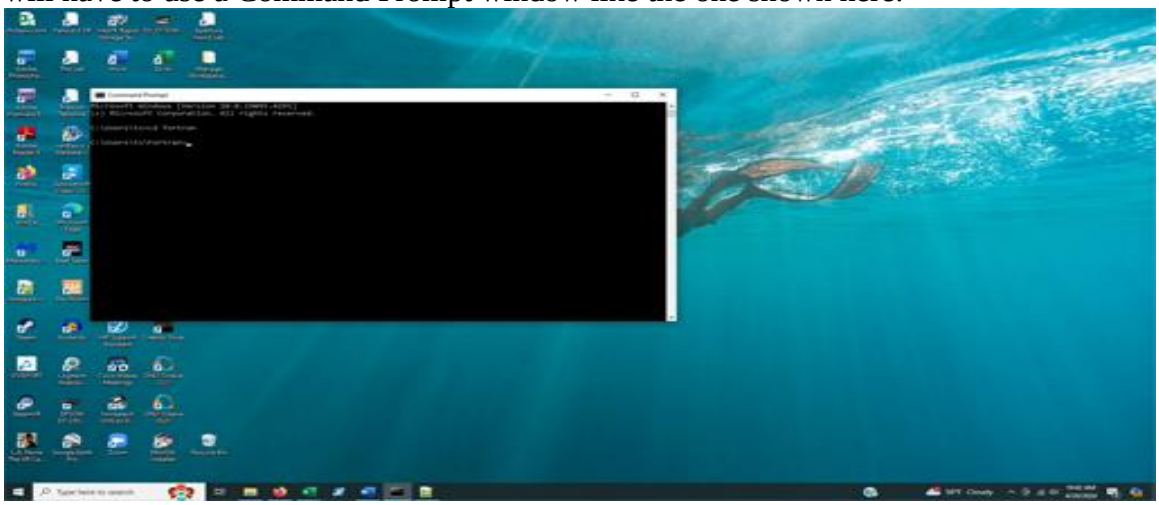
Click the Download button. Find the new …setup.exe file in your downloads folder. You will find your downloads folder in Windows File Explorer. Run the setup program and follow the instructions. If you followed all of the default options, you now have your compiler in a new folder, C:\MinGW.

To use the compiler, you will need to do one more thing. Click that extreme lower left icon on your desktop screen and choose “Settings” from the popup.



In the Settings window that opens, use the search field to type “environment”. You will get as far as “env” and the search results will already show what you want – select “Edit the

system environment variables". In the Advanced tab of the System Properties dialog that opens up, click Environment Variables. In the next dialog, select the User variable, Path, and click Edit… . Next, click New and type in
C:\MinGW\bin
"OK-your-way" back out and you have now added to your system's default paths that can be used from a Command Prompt window. Whoa! What's a Command Prompt window?
You didn't want to buy the fancy computer programming development suite that does offer a lot of wonderful support and organization and capability, so you are doing it old-school. You will have to use a Command Prompt window like the one shown here.



Go again to that windows button at lower/left and left click. Scroll down the popup to reveal the folder called Windows System. Click that, to expand to see the choices that include "Command Prompt". Now you can left click that to open a new working window on your desktop or right click and choose to pin it to the task bar so that you won't need to do all these steps every time you want to open a command prompt window.

The command prompt window will present to you a line of text that is a path followed by a greater than sign,>. This sign is the official command prompt. The path shown left of the prompt symbol is where you are presently positioned in the folder tree structure. Your typed commands will appear after this > command prompt (mouse doesn't work here). Use the command, cd .. (that's a "cd" followed by a space and two periods) and you will move up in the path. Type dir at the prompt and the contents of the folder at your current location will be listed. Type cd, space, and a folder name from among the names showing after doing "dir", to move back down the path. By the way, "cd" stands for "change directory". It's a lot clunkier than Windows File Explorer, but you'll get used to it. There are lots of cool commands you can do from here but these two, cd and dir, are all we need.
Historical note: The command prompt is the way everyone had to work with their computer in the old MSDOS (Microsoft disk operating system) days before Windows.
We need to download another bit of software that will be extremely useful. Go to

https://notepad-plus-plus.org/

and then click to go to their Download page and click to Download their latest version. (I am using v8.6.5). During installation, check the option box to put the icon on the desktop. This is a wonderfully built text file reader and editor. Program source code files will need to be clean and simple without all the overhead of, say, a Word .docx that would invisibly include tons of info about your document like margin settings, etc., so you will want this simple editor instead of a more capable word processor like Word. And it's safe and free.

To review, at this point we have our 3D printer and filament. We have our compiler. We have our text editor. We have a command prompt window, and we know how to maneuver around files and folders in both Windows File Explorer and in the command prompt. We are ready to start using these things.

Let's use File Explorer to create a comfortable place to do our programming. Select C: drive and use the tree structure to maneuver your way down to the folder you want to work from. You can start as high as C: itself if you wish. Use the menus to add a New folder. I used the name Fortran and mine looks like this:
C:\Users\ts\Fortran
Mine, and likely yours too, is set up to be case-insensitive. We will make sure that all the files you create will be organized into this folder. Now open Notepad++. Select New from the icon toolbar or from the pulldown menus if Notepad++ didn't open with a new, unnamed blank file at the ready. On the first line of this new text document type 6 spaces and then starting in column 7 type the word, Program, a space, and a name you make up as the name of your first program. In Fortran you will always skip the first 6 columns, as these are reserved for special use. This is not optional. Your program will not compile if you write commands in these first 6 spaces. Use the first 5 only for writing a number if you want to have a numbered line of code you may later want to branch to. Don't use column 6.
On the second line type IMPLICIT NONE. I use caps for emphasis because I strongly encourage you to use this optional instruction to the compiler, and also because caps is kind of a convention in Fortran, although not demanded. The last two lines of your program will be STOP and END. Do a carriage return; i.e., press Enter at the end of the line of text. Now use the mouse and/or the up/down/left/right keyboard arrow keys and insert lines after the IMPLICIT NONE line. But first, let's learn a new thing. Any text that follows an exclamation mark, !, will be invisible to the compiler and will serve as commentary for you to keep your code clear and understandable. So, after END type a space ! space and write your program's name. This will be helpful if it ends up a hundred lines below the top and is followed by subroutines (supporting code called by the main program), and while scrolling through your file, you need to know which END you are looking at! Now let's go back to adding new lines after IMPLICIT NONE.

Since we declared Implicit None, we must declare all variable names that we want to use. So we make up some names and define them. Add lines for variables X, Y, DIST, I, and A. With these we will demonstrate some common variable types.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*6 A
! More code is going to go in here
      STOP
      END ! SIMPLEMATH
```

Your program should look something like that above. “Reals” are floating point numbers stored in the computer as a bunch of digits and a power of ten. Integer numbers are treated differently and must be exact whole numbers. Character variables are for text strings, and while Fortran is considered clunky for working with text, it can still do marvelous things if you are clever. Now we are set up to add some lines to actually do something. Use “=’ to create an assignment of a value to be put into the designated variable location.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*6 A

      X = 5.1
      Y = 2.4721
      DIST=DSQRT(X*X + Y*Y)

      STOP
      END ! SIMPLEMATH
```

The square root intrinsic function is invoked here. Fortran has a lot of intrinsic functions that you may access. The basic function is SQRT(), but we use DSQRT() as the double precision version that is also available. (Real*8 variables are double precision, using twice the memory space as Real*4 single precision variables. I recommend double precision for everything because modern computers have lots of memory capacity and all of your math will be more accurate.)
We could now compile and run this program, and it would indeed do these operations, but we would watch the screen and see nothing since there is no I/O specified (Input/Output). So let’s add a print-output-to-screen capability.

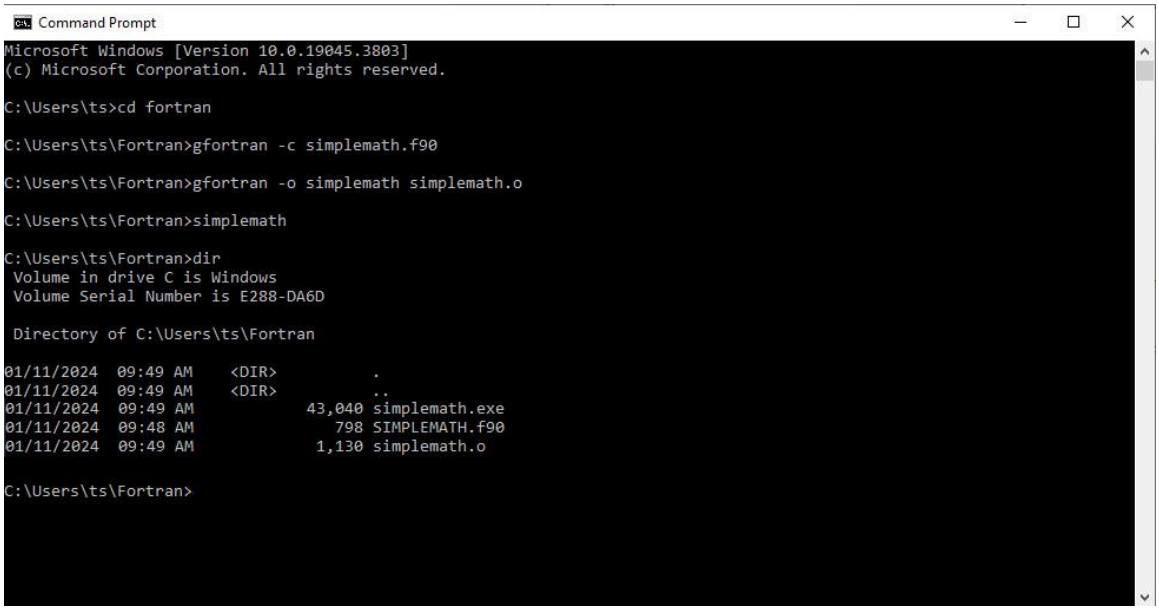
```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*6 A
      X = 5.1
      Y = 2.4721
      DIST=DSQRT(X*X + Y*Y)

      WRITE(*,*)DIST

      STOP
      END ! SIMPLEMATH
```

We could spend time learning lots more functions and nuances of formatted read and write statements, but instead at this point we will lurch ahead so that no one loses patience. In Notepad++ do a Save or a Save As… (same result if the file is new and as yet unnamed).

Give the new file a name with the extension .f90, say,SIMPLEMATH.f90. You have created a computer program. It must be compiled before it will run on your computer.
Now open a command prompt window and cd your way to your Fortran working folder that you created earlier with File Explorer. At the prompt, type
gfortran -c simplemath.f90
You have just invoked the compiler program which is that .exe you installed in MinGW, and it worked because you put the path C:\MinGW\bin into your system environment.
Your prompt just came back and you didn't see anything happen. Type dir and you will see that, in fact, a new file named simplemath.o is now there. This is an "object file". It is hexadecimal and you can open it in Notepad++, but it will look let garbage to us humans, so don't bother. Instead do the next step in the Command Prompt window. Type
gfortran -o simplemath simplemath.o



The dash-o and dash-c after the gfortran command are called switches. Shut up, sit down and just use the switches I give you! If you want to become a geek, you can learn on your own all the switches that control compiler options (I don't know them). Of the list of things here that follow the switches, the first after -o is the name you want to give your output file. You get to make it up. When that file is created by the compiler, it will appear with the .exe extension. You could have used any name, but I will be less confused if we use the same name for source code, object file and executable. You entered the command that will commence the "link" operation (more about this later), and again you see nothing happen, but again you could type dir and see that you have yet again another new file as your new executable was created. Don't bother with the dir, though. The next thing you may do is to just type the name you made up for your program. Type at the new prompt>
simplemath
Typing the name of the executable at the command prompt made your program run (typing the extension, in this case .exe, is not necessary). Alternatively, you could find your new executable in Windows Explorer and double click it, but your output window will open and close faster than you can see it, so let's run Simplemath from our open command window. Finally, a result appears. Look at all those double precision digits! You have just solved

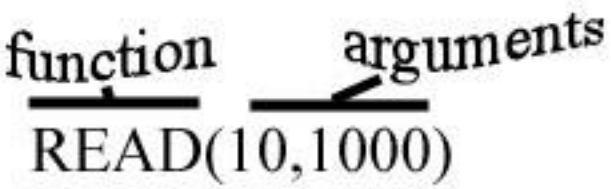
pythagoras's theorem for the diagonal length of a 5.1 by 2.4721 right triangle. Or equivalently, this is the distance from the point (0,0) to the point (5.1, 2.4721). Hurry! Send off your resume as a computer programmer to get that high paying job!

To do useful work (the kind we will want for driving a 3D printer) we are now going to add more programming skills to our capabilities. To preview, they are

1. "Unit numbers" and accessing files for I/O (input and output)
2. Subroutines
3. Common blocks
4. Conditional branching
5. Looping
6. Arrays

Our first program is a bit lame mainly because you can't specify the values of X and Y without editing the source code and recompiling. Note for the record: "compiling" is the step that created the object file, simplemath.o. The next step that makes the final product is called "linking" (the command that used the -o switch.) An advanced programmer will have multiple object files to be linked at this step. Your MinGW suite that you downloaded includes things called libraries that have functions like that SQRT() function we used in our program. The -o link operation will find the required support code and fold it into our program so that it can stand alone and work wherever we put it.

Now back to the topic at hand, getting values into X and Y. I/O is done to and from what are called "unit numbers." You can assign a unit number to a file in computer memory or you can use the unit numbers of existing peripherals (keyboard and monitor). Usually the keyboard is unit number 5 and the monitor (your screen) reserves unit number 6 for itself. Try not to use 5 or 6 yourself for any of your I/O files. In fact I stay away from all single digit unit numbers, scared that I might step on someone's toes. I vaguely remember from college 50 years ago unit 3 being reserved for some device – maybe punch card reader? I like to use unit 10 and up to be safe. We can write the program to read input from the keyboard during runtime with the READ statement and addressing unit 5. Better still is to use a star with the READ statement. When you use a star, the compiler is smart enough to find the keyboard just in case your program ends up on some computer that uses a number other than 5 for the keyboard input. Similarly, use a star for writes to the screen. FYI, the term "argument" refers to parameters in a list inside the parentheses that inform the routine being called.



For a read statement, the first argument would be the unit number to read from, and the second would be a formatting reference. Note that the arguments are separated by commas. The second argument after both the read and write commands shown here can be a number that refers to a format line in the source code. The star that we will use for now instead of a reference number means that this is unformatted so defaults will prevail. We will use a format line later. Now you understand the write statement and its syntax a little better.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*1 A
!     X = 5.1
!     Y = 2.4721
      READ(*,*)X,Y
      DIST=DSQRT(X*X + Y*Y)
      WRITE(*,*)'The answer is ',DIST
      STOP
      END ! SIMPLEMATH
```

Note: The term “syntax” refers to the precise structure including commas, parentheses, spacing, etc. that must be used in a particular line of code.

Notice new things above. I commented-out the old assignment statements with the “!” in column 1. These two lines are unnecessary as we are about to clobber their values with the read statement. It is kind of nice that we can see the evolution of our program code by leaving the X and Y assignment statements for us to see, while they will be invisible to the compiler because of that exclamation point. I also added a character string denoted by the use of single apostrophes in the write statement. The unformatted write statement lets you send out any number of text strings or variable values separated by commas.

Compile and link the above program. You would do well to discover that you can use the keyboard up arrow while in the command prompt window to reload prior commands from your session history (everything you’ve done since you opened this command prompt window.) So two quick up-arrow keystrokes, and you’re looking at your compile command again. Run it, then one up-arrow brings you right back and one down-arrow will show the most recent command that followed that one, which is your link command. One more up-down-enter and you’ve launched the newly compiled version of your program. Superfast. Almost getting fun yet?

Run your simplemath program repeatedly and note how generous that unformatted read(*,*) is. To type in your input, you can use decimals or not (3. or 3.0 or 3 are all the same), and you can put X and Y in at the same time separated by a space or a comma or who knows what? Or send X first and the computer will sit there and wait for the Y to come next.

We will do file I/O (as opposed to keyboard & screen) in a minute, but first let’s keep the keyboard and screen I/O in the code (our READ and WRITE statements) and learn conditional branching and looping.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*1 A
      A='Y'
      DO WHILE(A=='y'.OR.A=='Y')
        WRITE(*,*)'Enter the values for X and Y'
        READ(*,*)X,Y
        DIST=DSQRT(X*X + Y*Y)
        WRITE(*,*)'The answer is ',DIST
        WRITE(*,*)'Do you want to do another?'
        READ(*,*)A
      ENDDO ! WHILE(A=='y'.OR.A=='Y')
      STOP
      END ! SIMPLEMATH
```

DO WHILE is a Fortran function your compiler has in its library available for you to use. The lines of code between the DO WHILE and its corresponding ENDDO will be executed repeatedly in a loop until the condition in parentheses evaluates as false. The double equal sign is used to test for equality (recall that a single equal sign in a line of code is for the assignment statement, putting a value into a variable name.) Note that the logical test here uses the .OR. operator to combine two tests into the logical condition, testing for upper or lower case. I cleverly did this in case I stupidly still have my caps lock on after all that Fortran typing. Any response to the "do another" query other than Y or y will cause the execution to exit the loop. Note also that I assigned 'Y' to the character string variable, A, before the DO WHILE so that I wouldn't fail the test condition before the loop even got started.

Now we are ready to start reading input from a file and writing output to a file. Let's take the DO While loop back out so that we can get data from a file instead of waiting for keyboard input. Use Notepad++ to create another new text file. Remember how simple that was? Click the far left toolbar icon for a new file, or use the File pull down menu to select New, or on the keyboard type ctrl-N (don't use the Shift key, just hold the ctrl key while you press the keyboard N key.) Your new blank file is now open. You might type a helpful note on the first line to remind you of what this file is for, every time you open it and see it in your editor. Then start loading in X and Y data for your simplemath program to use. You can start typing in column1.

```
This file holds the X and Y input values for my program, SIMPLEMATH
3.3 8.9
9 5
4.5 6.7
22 11.8
```

Save the file like you did before with a name of your choosing and the .txt extension. I'm using INVARS.txt. The following code will make our program open and read INVARS.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X,Y,DIST
      INTEGER*4 I
      CHARACTER*1 A

      OPEN( 10, FILE='INVARS.txt', STATUS='OLD' )
      READ(10,*)A ! Read off that label line of text that we put first
100   CONTINUE
        READ(10,*,END=101)X,Y
        DIST=DSQRT(X*X + Y*Y)
        WRITE(*,*)'The hypotenuse of the right triangle is ',DIST
      GO TO 100
101   CONTINUE
      CLOSE(10)

      STOP
      END ! SIMPLEMATH
```

So, what is new here? We picked the number, 10, to attach to our input file and used the OPEN statement. We can now read, write or do a number of other operations on unit 10, which has been attached to the file INVARS.txt. The third argument of the OPEN statement here declares STATUS='OLD' so the file must exist in the same folder as simplemath.exe when you launch it. If you haven't made INVAR.txt yet, the open statement will fail and the execution will error-out. We have also introduced the READ statement with a third

argument, END=101. Note that this code now has numbered CONTINUE lines, 100 and 101. These line numbers are any numbers you want them to be, and they must be written in columns 1 through 5. We just picked 100 and 101 at random. Technically, we could have labelled the READ(…)X,Y with the 100 and not used the CONTINUE line (CONTINUE does nothing), and the program execution would be identical; i.e.,

```
100   READ(10,*,END=101)X,Y
        WRITE(*,*)'The hypotenuse of the right triangle is ',DIST
      GO TO 100
101   CLOSE(10)
```

However, it is good practice to have clean, dedicated entry points for looping and branching operations in your code. The END=<line no.> specification allows SIMPLEMATH to keep reading lines of INVARS.txt until it runs out of data, at which time execution will branch out of the GO TO loop.

Modern, professional programmers are at this point horrified that I am teaching GO TO's, but you may learn more sophisticated (e.g. C++) programming in the future. This instruction will use the methods easiest for the beginner to comprehend.

Another thing to note about this example is that the variable, A, was declared as only 1 character in size. The first READ statement in the code before the loop starts will put "T" from line 1 of INVARS.txt into A, and that's all we need to do to read past that first text line (that we put in as kind of a label for us to see when viewing the file in the editor.)

Looking back, we have learned quite a lot now, but let's add one more ability, the formatted write. Replace the * in the WRITE statement argument with a line number and create a format line in your code with that number. It looks like this.

```
        WRITE(*,2001)'The hypotenuse of the right triangle is ',DIST
2001    FORMAT(A40,F7.3)
```

The A40 designation provides room in the output file for a character string exactly 40 characters long for the first element in the write statement (The text that we wrote happens to be 40 characters long so we accommodated it with A40 in the WRITE statement's format), and then the F7.3 allocates a "field size" for the value of the second element, the value of the variable, DIST, of 7 spaces and it will include 3 digits to the right of the decimal point. So there will be room for up to 3 digits to the left of the decimal point (3 digits on the left plus the 3 to the right and 1 allotment for the decimal point makes 7.) An attempt to write out a value of 1000 or more will result in ***.*** showing up in your output. So, know your expected range of output and make the field big enough. If, say, you will be outputting a value like 2398.34857, you had better use at least F8.3 that will show 2398.349 or F7.2 that will show 2390.35 (you get the idea, nothing too deep here.)

Next, let's write the output to a file, instead of to the screen. Another OPEN statement will create an output file. This time the status is declared as unknown. This way the file may already exist in which case it will be overwritten, or if it doesn't exist, it will be created. Note: Sometimes you might want to use STATUS='NEW' instead of 'OLD' or 'UNKNOWN' as this will demand that the file does not already exist, thus protecting an existing file of that name. In such a scenario where the desired new file name already exists, the execution would error-out. Here is the new guts of SIMPLEMATH with unit 11 output and status unknown.

```
      OPEN(11,FILE='OUTVALS.txt',STATUS='UNKNOWN')
      OPEN(10,FILE='INVARS.txt',STATUS='OLD')
      READ(10,*)A ! Read off that label line of text that we put first
100   CONTINUE
        READ(10,*,END=101)X,Y
        DIST=DSQRT(X*X + Y*Y)
        WRITE(11,2001)'The hypotenuse of the right triangle is ',DIST
2001    FORMAT(A40,F7.3)
      GO TO 100
101   CONTINUE
      CLOSE(10)
      CLOSE(11)
```

Now, when you run this version, you see nothing return to the screen, but go to Notepad++ and do Open… to get the open dialog window, and you will find your new file exists, and it has all your lines of output as expected.

Now, let's separate the read and write operations into their own blocks of code (their own loops) and thus introduce the concept of arrays to hold data. First declare the X, Y, and DIST variables to be arrays. A size of 100 will be more than big enough. Change the REAL*8 line.

```
      REAL*8 X(100),Y(100),DIST(100)
```

You have now declared that X, Y, and DIST will actually be 100 variables each, instead of just 1, and can be referred to with an index. We provide that index as an integer inside of parentheses following the variable name, and address it like

```
      X(1)=3.3
      X(2)=9.
```

Better yet, use that integer variable, I, that we put in the original code and are now ready to put to work. Note that every time you execute the READ statement, the computer moves a pointer 1 step farther down the file content.

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X(100),Y(100),DIST(100)
      INTEGER*4 I,IR
      CHARACTER*1 A
      OPEN(11,FILE='OUTVALS.txt',STATUS='UNKNOWN')
      OPEN(10,FILE='INVARS.txt',STATUS='OLD')
      READ(10,*)A ! Read off that label line of text that we put first
      I=1
! Load up all the input data
100   CONTINUE
        READ(10,*,END=101)X(I),Y(I)
        I=I+1
      GO TO 100
101   CONTINUE
      IR=I-1 ! Save the count of records read
      CLOSE(10)
      WRITE(*,2002)IR
2002  FORMAT(I4,' records were read in')

! Now operate on the data
      DO I=1,IR
        DIST(I)=DSQRT(X(I)*X(I) + Y(I)*Y(I))
        WRITE(11,2001)DIST(I)
      ENDDO ! I=1,IR
2001  FORMAT('The hypotenuse is ',F7.3)
      CLOSE(11)
      STOP
      END ! SIMPLEMATH
```

Let's look at the new features we have added. There are, of course, the array variables. We also created a new variable name, IR, so we can save the number of lines of data that were successfully read. There is this new thing, the DO loop. We have the DO statement paired with the ENDDO statement, and any lines of code we want executed in between. It's pretty obvious that this loop is going to execute the lines within, incrementing the value of I until it reaches IR. Technical note: Just in case you run across it, there is also such a thing as a FOR loop that is very similar, but I am not going to bother explaining the nuanced differences here. Finally, you might notice that structurally we have separated the Read and Write operations with this new implementation. We can do the read a bunch of times, and later can do the write a bunch of times. We are now one step away from using subroutines.

The C++ guys are sneering, but I'm going to teach you the simple world of subroutines and common blocks. Those guys use things called prototypes and classes which are much harder to learn, but essentially do the same thing. Use a subroutine when there is a sequence of operations you want to do repeatedly from multiple places within the program. This will keep things clean and understandable rather than repeating great chunks of code throughout the source code file.

All the same rules (syntax, available intrinsic functions, etc.) apply inside of subroutines, the only exception being that line 1 instead of "Program X", will look like the following where we made up the name SUBX. (You will be able to use any name.):

```
      SUBROUTINE SUBX(arg1,arg2,…etc.)
```

There are two options for writing our subroutine: 1) we can start a new file in the editor and write it there, in which case, it will have to be saved and compiled into one of those .o object files ready to be named and linked during the gfortran command in the command prompt window, or 2) we can include the subroutine lines of code right within the file that has our main program and it will be compiled right along with the main program when it is compiled. We choose the latter! We invoke or "call" the subroutine from the main program with the CALL statement. Now we have replaced the embedded operations in the main code, with calls to subroutines. Make up the names LOADDAT and CALCDIST. (Or you can make up your own names.)

```
      Program SIMPLEMATH
      IMPLICIT NONE
      REAL*8 X(100),Y(100)
      REAL*8 DIST(100)
      INTEGER*4 I,IR
      COMMON /DTACOM/X,Y,IR

      CALL LOADDAT
      DO I=1,IR
        CALL CALCDIST(X(I),Y(I),DIST(I))
      ENDDO ! I=1,IR

      OPEN(11,FILE='OUTVALS.txt',STATUS='UNKNOWN')
      WRITE(11,*)'The DIST values for INVARS are as follows'
      DO I=1,IR
        WRITE(11,2000)I,DIST(I)
      ENDDO ! I=1,IR
      CLOSE(11)
2000  FORMAT(I4,F7.3)
      STOP
      END ! SIMPLEMATH

      Subroutine LOADDAT
      IMPLICIT NONE
```

```
      REAL*8 X(100),Y(100)
      INTEGER*4 I,IR
      COMMON /DTACOM/X,Y,IR
      CHARACTER*1 A
      OPEN(10,FILE='INVARS.txt',STATUS='OLD')
      READ(10,*)A ! Read off that label line of text that we put first
      I=1
100   CONTINUE
        READ(10,*,END=101)X(I),Y(I)
        I=I+1
      GO TO 100
101   CONTINUE
      CLOSE(10)
      IR=I-1
      RETURN
      END ! LOADDAT

      Subroutine CALCDIST(XC,YC,D)
      IMPLICIT NONE
      REAL*8 XC,YC,D
      D = DSQRT( XC*XC + YC*YC )
      RETURN
      END ! CALCDIST
```

You may think that having CALCDIST is no great advantage, and indeed here, in this example, it is not. But your future programs will have much longer, more complex calculations. The example presents two alternatives for communicating with subroutines.

First is the COMMON block that we use here with LOADDAT. Under this strategy, we are designating a location in computer memory for the indicated variable set with the common block named, DTACOM. Both the main and any subroutine that includes this common block can access these variables by the names listed. They are in fact the exact same place, so anything one routine does to them, the other "sees".

The other protocol for communicating with your subroutine is to provide an argument list. In the example, CALCDIST does this by defining 3 variables to be provided to it whenever it is called. The variable types have to agree (real, integer, character, etc.) between the subroutine and the calling statement or you will get some crazy errors. Under this second protocol, the subroutine has its own unique variables and its own variable names, just make sure those declared variable types agree! Note that your program can send the value of the real array variable with one specified index to a subroutine's non-array real variable. Here the call to CALCDIST is sending out the values of X(I) and Y(I) and receiving a value in DIST(I).

You now have sufficient knowledge to build very complex fortran programs. Oh, wait! There is one more thing that should be included in your basic tool set. The conditional branch.

```
      IF( <condition> ) <statement>
```

And

```
      IF( <condition> ) THEN
         <statements>
      ELSE
         <statements>
      ENDIF
```

You can guess these are going to be very useful. We have already been introduced to the "condition" concept with the DO WHILE above (note the syntax with the dots either side of the OR). Have fun building complex conditions using .AND. and .OR. and .NOT. and parentheses. The ELSE condition is optional. The THEN…ENDIF structure is also optional if you have only a single statement to execute on the IF(<condition>) that can be included on the same line of code. Here is an excerpt of code that presents an example of a use of IF…THEN…ENDIF:

```
REAL*8 THETA,THETACLIPPED
REAL*8 PI
DATA PI/3.14159265d0/
CALL SUBROUTINETHATDOESABUNCHOFSTUFF(THETA)
IF( THETA>2.d0*PI ) THEN
   THETACLIPPED = THETA - 2.d0*PI
ELSEIF( THETA<0.d0 )THEN
    THETACLIPPED = THETA + 2.d0*PI
ELSE
   THETACLIPPED = THETA
ENDIF
```

Presumably, you have written `SUBROUTINETHATDOESABUNCHOFSTUFF` that is calculating a value for the variable, `THETA`. The IF… construct here constrains `THETA` to be always positive and between zero and two PI. I sneaked in some new stuff just now. You can see how inside of the IF…THEN structure, you can have an ELSEIF…THEN as well as an ELSE… I also introduced the DATA statement. If you have a variable name you want to use and have a fixed value throughout your program or subroutine, declare it with the

DATA varname/value/

syntax that I show here. Using a DATA statement is likely not necessary, but is kind of nice for logical, clean code. Finally, when using real numbers in your double-precision code, append that `d0` after the digits to assure the computer clears non-zero trash that may lurk in memory in the far, right digits places. So write actual values 3.14159265 and 2. as 3.14159265d0 and 2.d0 to take full advantage of our double precision calculations.

OK, I'm done. If you want more, a useful jumping off point for further exploration is

https://gcc.gnu.org/wiki/GFortran

But we are not really done. I still promised you would be able to use your programming skills to drive your 3-D printer. Let me give you a complete subroutine I have written that converts any X, Y value you may calculate in your program into a line of gcode. Gcode is the syntax that you can feed to your printer (or other similarly numerically driven machinery). You are going to run your program. It will calculate your desired X and Y values, generate gcode equivalents and write them to a file. Then, having a cheap, not wifi connected printer, you will copy the file onto a micro SD memory card and carry it over to the printer. Tell the printer to process that file name, and voila, your product emerges. Details about the actual printing operation will be presented later.
You may research gcodes on the internet to find more, but right now I am introducing only G0 and G1. A G0 command will move the print head (no filament extrusion), and G1 will draw with the print head (extrude filament while moving.)

```
      SUBROUTINE WRITEOUT(X,Y,MD)
      IMPLICIT NONE
!     WRITEOUT SENDS X AND Y TO THE PLOTTING FILE UNIT 14 AND TO THE GCODE FILE UNIT 15
!     MD 0 OR 1 WILL FLAG MOVE OR DRAW
!     LAYER INCLUDED IN THE COMMON BLOCK LETS YOU SELECT AND LIMIT THE PLOT FILE DATA
      REAL*8 X,Y,X110,Y110,XOLD,YOLD,EVAL,ERATE
      COMMON /PRIORXY/XOLD,YOLD,ERATE,EVAL
      INTEGER*4 LAYER,PLOTLAYER
      INTEGER*4 MD
      COMMON /LAYERCOM/LAYER,PLOTLAYER
      CHARACTER*80 A
      CHARACTER*10 XCH,YCH,ECH
      INTEGER*4 YSTRT,ESTRT
      REAL*8 DIST
      IF(MD==0)THEN
        A='G0 X                                                                    '
      ELSE
        A='G1 X                                                                    '
      ENDIF
!     ACKNOWLEDGE THAT 110,110 is the center of print area
      X110=X+  110.D0
      Y110=Y+  110.D0
      IF(LAYER==PLOTLAYER)WRITE(14,1000)X110,Y110
      CALL RTOC(X110,XCH,1)
      CALL RTOC(Y110,YCH,1)
      A(5:14)=XCH               ! (xx.xxx    ) or (xxx.xxx   )
      YSTRT=13
      IF(A(11:11)==' ')YSTRT=12
      A(YSTRT:YSTRT)='Y'
      YSTRT=YSTRT+1
      A(YSTRT:YSTRT+9)=YCH      ! (xx.xxx    ) or (xxx.xxx   )

      IF(MD==1)THEN ! A DRAW (as opposed to a MOVE) WILL NEED TO PUSH SOME FILAMENT
        DIST=DSQRT((X-XOLD)*(X-XOLD)+(Y-YOLD)*(Y-YOLD))
        EVAL=EVAL+DIST*ERATE

        CALL RTOC(EVAL,ECH,2)
        ESTRT=YSTRT+8
        IF(A(YSTRT+6:YSTRT+6)==' ')ESTRT=YSTRT+7
        A(ESTRT:ESTRT)='E'
        ESTRT=ESTRT+1
        A(ESTRT:ESTRT+9)=ECH
      ENDIF ! MD==1

      WRITE(15,1004)A
      XOLD=X
      YOLD=Y
1004  FORMAT(A80)
1000  FORMAT(F9.3,', ',F9.3)
      RETURN
      END !WRITEOUT
```

You have learned everything you need to know to understand what SUBROUTINE WRITEOUT is doing. Presume that you have a main program that is calculating your X and Y values, and for each calculation, it is also then doing a CALL WRITEOUT(X,Y,0) or CALL WRITEOUT(X,Y,1) depending on whether you want filament extrusion or not. This routine has the added bonus that in addition to constructing the proper Gcode to go to the print file, it simultaneously puts those values into a comma delimited text file that Excel can open and plot to show your intended tool paths. This is extremely useful to assure that you are getting what you want before sending crazy stuff to the printer!

See here how WRITEOUT uses both argument list and common block to know the values coming from main (i.e., your "main" calling program.), like the filament rate, the current layer being printed, and the chosen layer to plot. You may also see how the routine can address individual places in a character variable's string with the parentheses and the colon while it carefully constructs the correct character string to output. Finally, you probably saw how the subroutine calls another subroutine, RTOC. I need to give you that one too. In fact, I am going to give you an entire example 3D-printer-gcode-file-generating example below. Note how RTOC puts a lot of effort into tailoring exactly how the line of Gcode is going to look. We want it to be pretty and not confuse the printer.

Subroutine RTOC uses a unit 23 (a number we just chose out of thin air.) We declare unit 23 in main with a kind of STATUS ("scratch") you haven't seen yet. I put the statement to Open unit 23 as "scratch" in the main program to open it just once and then close it at the end of main. RTOC will use unit 23 repeatedly while the program executes. Let's see what else is new in the example you are about to read below. Well, actually, not much. While the example is very complicated, it employs only the programming tools you now already know. Wait, I can find one thing in this code that I haven't yet introduced. The DBLE() function is applied to integer values to do a proper conversion of our integer variable to a double precision real number in computer memory for those cases where we are doing real and integer mixed math. Your compiler might let you get away with mixed math, but I think the places where we use DBLE(LAYER) and DBLE(I-1) in the example are important for accuracy.

Now, lets take it from the top and do an entire project. Trial and error with my machine and my filament have taught me that the following parameters work well with these values:

Nozzle temp	210
Bed temp	59
Print speed	2100
Erate	.032
Layer0 Erate	.059
Spacing between passes to fuse	.35

Our example program starts by loading required lead-in lines into the gcoded print file, establishing the above chosen values for temperatures, speed, etc. An enhancement not offered here is to do a user interface (write prompts to the screen and reads from the keyboard) that allows you to change these values at run-time. I'll put in code to let you adjust one of the parameters, PLOTLAYER, to show how it's done. After creating the startup code lines (you can research on your own what the various M-codes and G-codes mean), I then typically generate lines to run the nozzle at zero height for a short distance to clean the tip and then do the usual skirt to "warm-up" the flow. Normal is a round skirt, but I'd have to teach you the G3 or G2 command, so here we do just a square box. We make it bigger than our pyramid object that we are going to make and skirt around the area to be printed on. This is essential to establish a healthy, steady flow as well as prove that your bed is level. Then the business of generating the toolpaths with each layer begins. Side note: viewing code in Notepad++ in a file with the .f90 extension will present fortran keywords, constants, comments, etc. color coded, making code nicely readable.

```
      Program SQBPYRAMID
! Build a file of gcode instructions to 3D-print a square based pyramid
      IMPLICIT NONE
      INTEGER*4 I,J,LAYER,NOL,EXITFLAG,PLOTLAYER
      REAL*8 BASE,BASEO,DB
      COMMON /LAYERCOM/LAYER,PLOTLAYER
      CHARACTER*80 A
      REAL*8 X,Y,Z
      REAL*8 XOLD,YOLD,ERATE,EVAL,NOME,NOMEIN,NOMEZERO
      COMMON /PRIORXY/XOLD,YOLD,ERATE,EVAL
      CHARACTER*10 LCH,ECH
      REAL*8 PI
      DATA PI/3.14159265d0/
      OPEN(14,FILE='XY.txt',STATUS='UNKNOWN')
      OPEN(15,FILE='SQBPYRAMID.gcode',STATUS='UNKNOWN')
      open(23,status='SCRATCH')
      A='                                                                                '
      X=0.d0
      Y=0.d0
      EVAL=0.d0
      NOMEIN=.032d0
      NOMEZERO=.059d0
      Z=0.d0
      NOL= 115 !number of layers
      PLOTLAYER=0
      EXITFLAG=0
      WRITE(*,*)'ENTER THE REQUESTED PLOT LAYER:' ! add this to enable you to select one
      READ(*,*)PLOTLAYER                          ! LAYER to graph in Excel

      CALL STARTUPGCODE

      LAYER=0
! MAKE A SKIRT BIGGER THAN THE OBJECT TO WARM UP THE NOZZLE FLOW
! BUT FIRST GIVE AN APPROPRIATE APPROACH FROM THE 0,0,0 STARTING POINT
! IN THE LEFT/FRONT CORNER OF THE PRINT BED
      WRITE(15,5)'G0 X170.000 Y50.000 Z0.100                        '
      WRITE(15,5)'G0 X40.000 Y50.000 Z0.100                         '
      WRITE(15,5)'G0 X75.000 Y75.000 Z0.300                         '
      WRITE(15,5)';TYPE:SKIRT                                       '
      IF(LAYER==PLOTLAYER)THEN ! FOR EXCEL PATH REPLICATION ONLY
        WRITE(14,1001)
        X=0.d0
        Y=0.d0
        WRITE(14,1000)X,Y
        X=170.d0
        Y=50.d0
        WRITE(14,1000)X,Y
        X=40.d0
        WRITE(14,1000)X,Y
        X=75.d0
        Y=75.d0
        WRITE(14,1000)X,Y
      ENDIF ! (LAYER==PLOTLAYER)
! Above is in absolute print bed coordinates (0,0) at front left corner
! From here on, X and Y values will be in the object's own coordinate
! system. The call to WRITEOUT is hardwired to add the 110,110 bias
! to center the object.

!     A SQUARE SKIRT, GO AROUND TWICE
!     and make the 2nd pass .35mm outside the 1st
      EVAL=0.d0
      ERATE = NOMEZERO
      XOLD=X
      YOLD=Y
      DO I=1,2
        X=-35.d0 - DBLE(I-1)*.35d0
        Y=-35.d0 - DBLE(I-1)*.35d0
        CALL WRITEOUT(X,Y,1)
        Y=Y+70.d0 + DBLE(I-1)*.7d0
        CALL WRITEOUT(X,Y,1)
        X=X+70.d0 + DBLE(I-1)*.7d0
```

```
        CALL WRITEOUT(X,Y,1)
        Y=Y-70.d0 - DBLE(I-1)*.7d0
        CALL WRITEOUT(X,Y,1)
        X=X-70.d0 - DBLE(I-1)*.7d0
        CALL WRITEOUT(X,Y,1)
      ENDDO ! I=1,2
      WRITE(15,5)';END SKIRT                                          '
      WRITE(15,5)';BEGIN OBJECT ZERO LAYER                            '

      BASEO=30.d0  ! PARAMETERS
      DB=.25d0     ! OF THE PYRAMID

      LAYER=-1
      DO WHILE(EXITFLAG==0)
        LAYER=LAYER+1
!       Put a LAYER note comment in the gcode file
        IF(LAYER.NE.0)THEN ! Don't relabel layer 0 because this was done above
         A=';LAYER:                                                                    '
         CALL RTOC(DBLE(LAYER),LCH,1)
         IF(LAYER<10)THEN
           A(8:8)=LCH(1:1)
         ELSEIF(LAYER<100)THEN
           A(8:9)=LCH(1:2)
         ELSEIF(LAYER<1000)THEN
           A(8:10)=LCH(1:3)
         ELSE
           A(8:11)=LCH(1:4)
         ENDIF ! LAYER<10 or else…
         WRITE(15,1004)A
        ENDIF ! LAYER .NE. 0
        Z=Z+.15d0
        IF(LAYER==0)Z=.30d0
        NOME=NOMEZERO ! TYP .059
        IF(LAYER.NE.0)THEN ! Don't rewrite Z at layer 0 because this was done above
         NOME=NOMEIN ! TYP .028
         A='G0 Z                                                                       '
         CALL RTOC(Z,LCH,1) !(x.xxx     ) or (xx.xxx    )
         IF(Z<10)THEN
           A(5:8)=LCH(1:4)
         ELSE
           A(5:9)=LCH(1:5)
         ENDIF! Z<10 or else
         WRITE(15,1004)A
        ENDIF ! LAYER .NE. 0
        EVAL=0.d0
        WRITE(15,1004)'G92 E0
'
        ERATE=NOME

!       IF(LAYER==0)LAYER=110 ! FAST SKIP AHEAD FOR DEBUGGING

        IF(LAYER<111)THEN ! shape 1
! $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
! ----------------------------- Shape 1 -------------------------------
! OVER THIS SPAN OF LAYERS, THE PERIMETER WILL GET SMALLER AS LAYER GOES UP
! MAKE THE WALLS BE 3 PASSES THICK OR APPROX 2 mm
          IF(LAYER==PLOTLAYER)WRITE(14,1001) ! column labels in Excel will separate data
          BASE=BASEO - DBLE(LAYER)*DB
          DO I=1,3
            X=-BASE/2.d0 + DBLE(I-1)*.35d0
            Y=-BASE/2.d0 + DBLE(I-1)*.35d0
            CALL WRITEOUT(X,Y,1)
            Y=Y+BASE - DBLE(I-1)*.7d0
            CALL WRITEOUT(X,Y,1)
            X=X+BASE - DBLE(I-1)*.7d0
            CALL WRITEOUT(X,Y,1)
            Y=Y-BASE + DBLE(I-1)*.7d0
            CALL WRITEOUT(X,Y,1)
            X=X-BASE + DBLE(I-1)*.7d0
            CALL WRITEOUT(X,Y,1)
          ENDDO ! I=1,3
```

```
        ELSE ! Shape 2
! $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
! ----------------------------- Shape 2 -------------------------------
! OVER THESE LAYERS, THE TIP IS CLOSED BY DRAWING A CROSS
          IF(LAYER==PLOTLAYER)WRITE(14,1001) ! column labels in Excel will separate data
          BASE=BASEO - DBLE(LAYER)*DB
          X=0.d0
          Y=-BASE/2.d0 + .7d0
          CALL WRITEOUT(X,Y,1)
          Y= BASE/2.d0 - .7d0
          CALL WRITEOUT(X,Y,1)
          X=-BASE/2.d0 + .7d0
          Y=0.d0
          CALL WRITEOUT(X,Y,1)
          X= BASE/2.d0 - .7d0
          CALL WRITEOUT(X,Y,1)
        ENDIF ! Levels

        IF(LAYER>=NOL)EXITFLAG=1
      ENDDO ! WHILE EXITFLAG (LAYER has finished to NOL)

      ! OPEN(10,FILE='gcodeEnd.gcode',STATUS='OLD')
! 200   CONTINUE
        ! READ(10,'(A80)',END=201)A
        ! WRITE(15,1004)A
      ! GO TO 200
! 201   CONTINUE
      ! CLOSE(10)
! SINCE YOU DON'T HAVE the file gcodeEnd.gcode to append, I will put its contents here
      WRITE(15,5)'M107                                                '
      WRITE(15,5)';End GCode                                          '
      WRITE(15,5)'M104 S0    ;extruder heater off                     '
      WRITE(15,5)'M140 S0    ;heated bed heater off (if you have it)'
      WRITE(15,5)'G91               ;relative positioning             '
      WRITE(15,5)'G1 E-2 F300       ;retract the filament a bit       '
      WRITE(15,5)'G1 Z+10 E-3 X-20 Y-20 F3600 ;Z & filmnt even more '
      WRITE(15,5)'M84                         ;steppers off           '
      WRITE(15,5)'G90                         ;absolute positioning   '
      WRITE(15,5)'M81                                                 '
5     FORMAT(A50)
1000  FORMAT(F9.3,', ',F9.3)
1001  FORMAT('X, Y')
1004  FORMAT(A80)
      close(23)
      CLOSE(14)
      CLOSE(15)
      STOP
      END ! SQBPYRAMID

      SUBROUTINE WRITEOUT(X,Y,MD)
      IMPLICIT NONE
!     WRITEOUT SENDS X AND Y TO THE PLOTTING FILE UNIT 14 AND TO THE
!     GCODE FILE UNIT 15
!     MD 0 OR 1 WILL FLAG MOVE OR DRAW
!     LAYER and PLOTLAYER INCLUDED IN THE COMMON BLOCK LETS YOU SELECT
!     AND LIMIT THE PLOT FILE DATA
      REAL*8 X,Y,X110,Y110,XOLD,YOLD,EVAL,ERATE
      COMMON /PRIORXY/XOLD,YOLD,ERATE,EVAL
      INTEGER*4 LAYER,PLOTLAYER
      INTEGER*4 MD
      COMMON /LAYERCOM/LAYER,PLOTLAYER
      CHARACTER*80 A
      CHARACTER*10 XCH,YCH,ECH
      INTEGER*4 YSTRT,ESTRT
      REAL*8 DIST
      IF(MD==0)THEN
        A='G0 X                                                       '
      ELSE
        A='G1 X                                                       '
      ENDIF
```

```
!     ACKNOWLEDGE THAT 110,110 is the center of print area
      X110=X+  110.D0
      Y110=Y+  110.D0
!     IF(LAYER==PLOTLAYER)WRITE(14,1000)X110,Y110,MD ! PLOT THIS LAYER ONLY
      IF(LAYER==PLOTLAYER)WRITE(14,1000)X110,Y110   ! choose not to put MD column XY.txt file
!     IF(LAYER>=PLOTLAYER)WRITE(14,1000)X110,Y110 ! SHOW MULTIPLE LAYERS FROM HERE TO THE END
      CALL RTOC(X110,XCH,1)
      CALL RTOC(Y110,YCH,1)
      A(5:14)=XCH               ! (xx.xxx    ) or (xxx.xxx   )
      YSTRT=13
      IF(A(11:11)==' ')YSTRT=12
      A(YSTRT:YSTRT)='Y'
      YSTRT=YSTRT+1
      A(YSTRT:YSTRT+9)=YCH      ! (xx.xxx    ) or (xxx.xxx   )

      IF(MD==1)THEN ! A DRAW (as opposed to a MOVE) WILL NEED TO PUSH SOME FILAMENT
        DIST=DSQRT((X-XOLD)*(X-XOLD)+(Y-YOLD)*(Y-YOLD))
        EVAL=EVAL+DIST*ERATE

        CALL RTOC(EVAL,ECH,2)
        ESTRT=YSTRT+8
        IF(A(YSTRT+6:YSTRT+6)==' ')ESTRT=YSTRT+7
        A(ESTRT:ESTRT)='E'
        ESTRT=ESTRT+1
        A(ESTRT:ESTRT+9)=ECH
      ENDIF ! MD==1

      WRITE(15,1004)A
      XOLD=X
      YOLD=Y
1004  FORMAT(A80)
!1000  FORMAT(F9.3,', ',F9.3,', ',I3)
1000  FORMAT(F9.3,', ',F9.3)
      RETURN
      END !WRITEOUT

      SUBROUTINE RTOC(A,C,I)
!     REAL number to CHARACTER conversion
      IMPLICIT NONE
      INTEGER*4 I ! Designate X,Y data or E data
      INTEGER*2 J
      REAL*8 A
      CHARACTER*10 C
      IF(I==1)THEN ! X,Y data with 3 digits to right of decimal
        IF(A<0.d0)THEN
          IF(A>-99.9995)THEN        ! -xx.xxx
            WRITE(23,2401)A
2401        FORMAT(F8.3)!(-xx.xxx )
          ELSE                      ! -xxx.xxx
            IF(A>-100.)A=-100. ! Catch the between rarity
            WRITE(23,2402)A
2402        FORMAT(F8.3)    !(-xxx.xxx)
          ENDIF
        ELSE ! A>=0
          IF(A<99.9995)THEN         ! xx.xxx
            WRITE(23,2301)A
2301        FORMAT(' ',F6.3,' ')!( xx.xxx )
          ELSE                      ! xxx.xxx
            IF(A<100.)A=100. ! Catch the between rarity
            WRITE(23,2302)A
2302        FORMAT(' ',F7.3)    !( xxx.xxx)
          ENDIF
        ENDIF ! A<0 OR NOT
      ELSE ! E data with 5 digits to right of decimal
        IF(A<9.999995)THEN          ! x.xxxxx
          WRITE(23,2303)A
2303      FORMAT(' ',F7.5)    !( x.xxxxx)
        ELSEIF(A<99.999995)THEN     ! xx.xxxxx
          IF(A<10.)A=10.
          WRITE(23,2304)A
2304      FORMAT(F8.5)        !(xx.xxxxx)
```

```
        ELSEIF(A<999.999995)THEN
          IF(A<100.)A=100.
          WRITE(23,2305)A
2305      FORMAT(F9.5)         !(xxx.xxxxx)
        ELSE
          IF(A<1000.)A=1000.
          WRITE(23,2306)A
2306      FORMAT(F10.5)        !(xxxx.xxxxx)
        ENDIF
      ENDIF
      BACKSPACE(23)
      READ(23,'(A10)')C
! Always read 10 characters. If it's X or Y or E<100, it has leading blanks
! Clean up the blank
      DO WHILE(C(1:1)==' ')
        DO J=1,9
          C(J:J)=C(J+1:J+1)
        ENDDO
      ENDDO
      RETURN
      END !RTOC

      SUBROUTINE STARTUPGCODE
! Build the starting Gcode
      WRITE(15,5)'M140 S59.000000                                    '
      WRITE(15,5)'M109 S210.000000                                   '
      WRITE(15,5)'M190 S59.000000                                    '
      WRITE(15,5)'G21            ;metric values                      '
      WRITE(15,5)'G90            ;absolute positioning               '
      WRITE(15,5)'M82            ;set extruder to absolute mode      '
      WRITE(15,5)'M107           ;start with the fan off             '
      WRITE(15,5)'G92 X0 Y0 Z0   ;Dont use G28 Home                  '
      WRITE(15,5)';               hand set to DEFINE AS ZERO instead'
      WRITE(15,5)'G1 Z15.0 F3600 ;move the nozzle up 15mm            '
      WRITE(15,5)'G92 E0         ;zero the extruded length           '
      WRITE(15,5)'G1 F200 E3     ;extrude 3mm of filament            '
      WRITE(15,5)'G92 E0         ;zero the extruded length again     '
      WRITE(15,5)'G1 F2100                                           '
      WRITE(15,5)';Put printing message on LCD screen                '
      WRITE(15,5)'M117 SQBPYRAMID                                    '
      WRITE(15,5)'M106 S255                                          '
      WRITE(15,5)';LAYER:0                                           '
5     FORMAT(A50)
      RETURN
      END ! STARTUPCODE
```

Presuming that you have used Notepad++ to put the above example problem code into a file named SQBPYRAMID.f90, you will compile and run SQBPYRAMID and see the two output files, XY.txt and SQBPYRAMID.gcode in your fortran folder. Use excel to open the XY.txt file and create a plot from the data (the plot data will be for whichever LAYER you chose to export when you ran the program.) Remember there is a label X,Y at the start of each set of data so select the data range for plotting accordingly. Examine the plot and assure that toolpath movements look reasonable. LAYER 0 will also include the approach and skirt, so it may be a little more interesting to look at. (Any one layer of this square pyramid is pretty boring.)

Now carry your gcode file to the printer on a memory card, plug the card in first, then power up the printer. My printer won't "hot swap" a card. It has to be there at startup! Use the user input screen to select temperatures for nozzle and bed and set them to the values your print is going to ask for. While these are warming up, you can make sure your bed is clean and ready. Then physically push the bed (Y-axis), the print head (X-axis) and roll the coupler at the bottom of the vertical Z-axis drive screw to position the print head at the left, front corner and down just touching the bed. This is tricky as the bed is spring balanced and you don't

want to deflect the spring as you set down. To this end, I like to gently slide my finger along the coupler to turn the screw and observe when the finger slides against the coupler without turning the screw further. You have made contact. Now grip the coupler firmly and back off a microscopic amount. If you get this right, your approach and skirt will look good. If too tight, the filament drive will buck. If too loose, the emerging filament will float around without sticking to the bed.

Back at the control panel, select Print From Media… and find and select your gcode file. There may be a delay yet for a few seconds before the axes move if the temperatures aren't quite yet stabilized. If things are going crazy, or you just didn't get that tricky starting bed clearance right, you can find a Stop Print command on the control panel screen. If you have done one of these Stop Prints, it's probably a good idea to cycle the power off and on before trying again.

Your print has finished and looks awesome! The drivers have let go of the stepper motors, so you are again free to physically move things. You may power down the printer or not. Do not scrape the object off while on the bed; you worked too hard leveling it to bang on the table. I presume you have a somewhat flexible surface pad that is clipped to the table top. Remove it and, away from the printer, use that scraper/spatula thing to remove the object and clean off skirt and initial (0,0) dribble, flexing the pad if needed.

Supporting information:

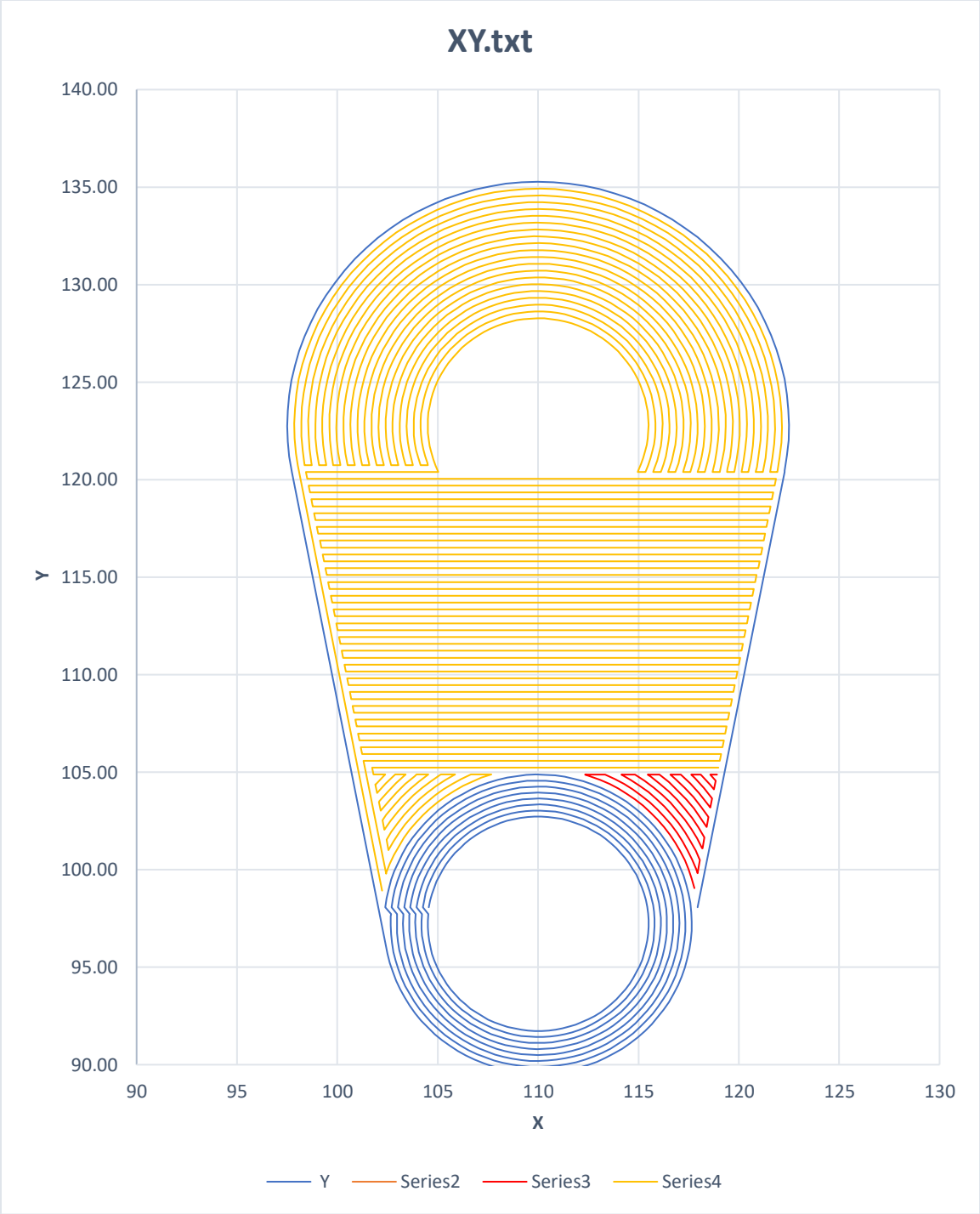
I'm not going to coach you through the assembly of the machine. It's fairly straightforward. Imitate the pictures and do what seems logical. I might advise building your own stand, off to the side (I made one out of wood) to hold the filament spool, since the two options for mounting, each, had drawbacks. Mounting on the side gets in the way and limits the vertical range, and mounting on the top gives a terrible feed angle into the filament drive.

For leveling, I like the thin piece of paper under print head sliding friction test. Get all 4 corners feeling the same by turning the adjustment wheels under the print table.

The only enhancement I did to my super entry-level printer, was to upgrade the plastic single gear E-drive (filament drive) to a flashy looking, metal, dual gear drive. It was remarkably cheap to do even when I had to get a new motor because the original motor's shaft was too short to support the new gear assembly.

I hope you enjoyed this short course and learned something useful. Here is a farewell look at the kind of things you can do. The following shows the tool path for one of the layers in the motorcycle turn indicator stem project. Note that I work at making the code's logic create well spaced print head passes that never cross. I also end the layer close to where the next layer is going to begin to minimize the dribble during moves. I had so much trouble with this dribble that I pretty much stopped trying to do moves (without drawing) or filament retractions. With my cheap printer, I get the flow going and never stop, choosing benign paths between printing areas. This is so much more fun than using CAD programs and "slicing software".

Questions? Use the Contact Us at dualminds.org and I will watch for your question. Drop the keyword, mindduel, and it will help me find your inquiry relevant to this tutorial.



The print order is the blue first, then the red, then the yellow. Placement of the Write(14,1001) statements in my Fortran code helps me use Excel to set apart the section of interest in the different color. The part of the path shown here in red is created by the following code. The code here calls my G3ARC subroutine that creates G3 and G2 gcode syntax for these arcs. The math uses the geometry of the intersection of a circle and a line

and thus invokes the Quadratic Equation that you may spy lurking in the code. The equation for a straight line is of type:

$$y = mx + b$$

(Sorry, I have not shown the calculation of "em" and "bee" variables in the code below)
And the equation for the locus of points on a circle is:

$$x*x + y*y = r*r$$

Most every point needed to be found comes from the algebraic calculation of intersections of these two equations. An excerpt of the code follows:

```
          IF(LAYER==PLOTLAYER)WRITE(14,1001)
! FILL RIGHT TRIANGLE
!         RP= will be 7.65 out of above loop
          RP=RP+.35d0
          bee1in= bee+bstep
          AQ=  1.d0 + 1.d0/(em*em)
          BQ= -2.d0 * (bee1in/(em*em) + YC1)
          CQ= bee1in*bee1in/(em*em) + YC1*YC1 - RP*RP
          Y = (-BQ + DSQRT(BQ*BQ-4.d0*AQ*CQ) ) / (2.d0*AQ) ! I/S RP AND 1 LINE IN
          X=(Y-bee1in)/em
          CALL WRITEOUT(X,Y,1) ! DRAW TO PT ON NEXT RP UP AND 1ST LINE INSIDE THE TANGENT
          YTOP=YC1+RP-.35d0
          XSIDE= (YTOP-bee1in)/em ! largest X at top of fill area
          J=0
!         DO WHILE(Y<YTOP.AND.X<XSIDE)
          DO I=1,11 ! arc and over
            J=1-J
            DIRARC=2+J ! 1ST TIME IS CCW=G3 (J=1)
            IF(J==1)THEN ! DO A CCW ARC, THEN MOVE OUT ALONG THE TOP
              JARC=-(Y-YC1)
              IARC=-X
              XTOP  = DSQRT(RP*RP-(YTOP-YC1)*(YTOP-YC1))
              CALL G3ARC(XTOP,YTOP,IARC,JARC,DIRARC)
              RP=RP+.35d0
              XTOP  = DSQRT(RP*RP-(YTOP-YC1)*(YTOP-YC1))
              IF(XTOP>XSIDE)XTOP=XSIDE
              CALL WRITEOUT(XTOP,YTOP,1)
            ELSE ! DO A CW ARC, THEN MOVE UP THE SIDE
              CQ= bee1in*bee1in/(em*em) + YC1*YC1 - RP*RP
              Y = (-BQ + DSQRT(BQ*BQ-4.d0*AQ*CQ) ) / (2.d0*AQ) !YSIDE
              X = (Y-bee1in)/em                                !XSIDE
              IARC=-XTOP
              JARC=-(YTOP-YC1)
              CALL G3ARC(X,Y,IARC,JARC,DIRARC)
              RP=RP+.35d0
              CQ= bee1in*bee1in/(em*em) + YC1*YC1 - RP*RP
              Y = (-BQ + DSQRT(BQ*BQ-4.d0*AQ*CQ) ) / (2.d0*AQ) !YSIDE
              X = (Y-bee1in)/em                                !XSIDE
              CALL WRITEOUT(X,Y,1)
            ENDIF ! (J==1) OR NOT
          ENDDO ! I=1,11
!         ENDDO !(WHILE)
          IF(LAYER==PLOTLAYER)WRITE(14,1001)
```

Fun facts about the tool path shown above:

I do a last wrap around to give the printed object a smooth exterior at this layer. Note in the photograph of the motorcycle part that over some ranges of layers, I chose to do this, and over other layers I left the left-right zig-zag path show through to the perimeter, giving that textured look.

The path is designed to never cross itself and ends very close to where it began. So the move to the next layer will have minimal dribble path.