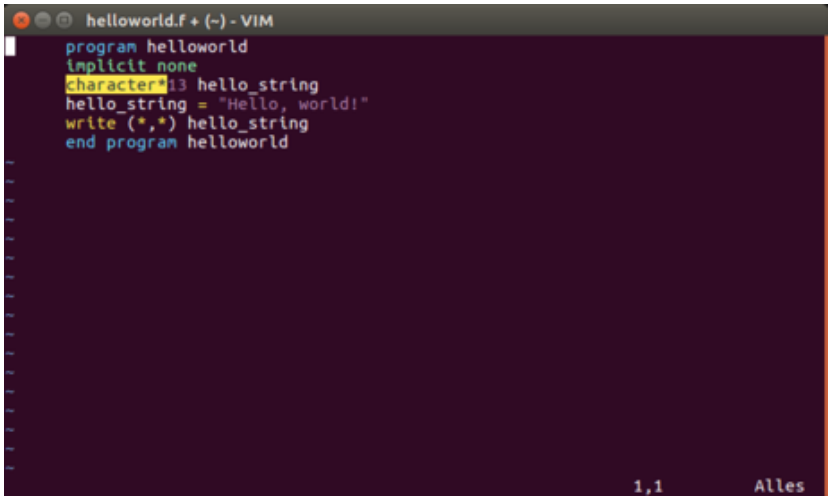
How to Program in Fortran

Many people perceive Fortran as an archaic and 'dead' programming language. However, most scientific and engineering code is written in Fortran. As such, programming in F77 and F90 remains a necessary skill for most technical programmers....

Part 1 of 4:

Writing and Compiling a Simple Program

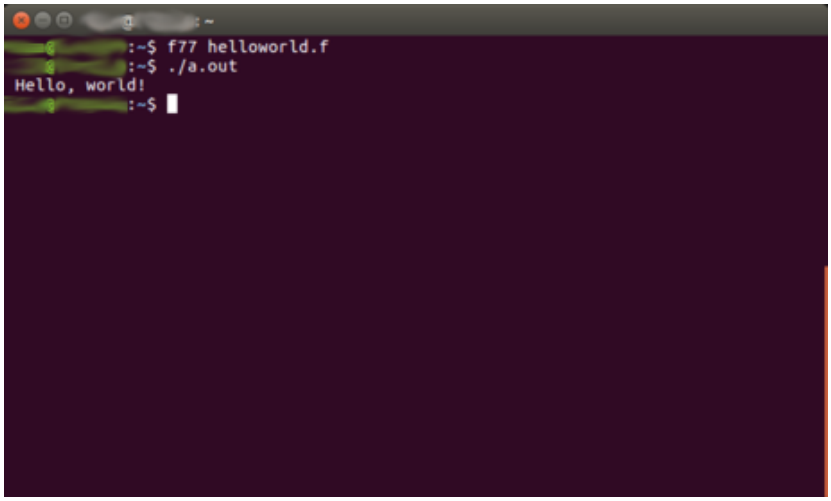
1. 

   Write a "Hello World" program. This is usually the first program to write in any language, and it just prints "Hello world" to the screen. Write the following code in any text editor and save it as `helloworld.f`. Pay attention that there must be *exactly 6 spaces* in front of every line.

```
 program helloworld implicit none character*13 hello_string
hello_string = "Hello, world!" write (*,*) hello_string end program
helloworld
```

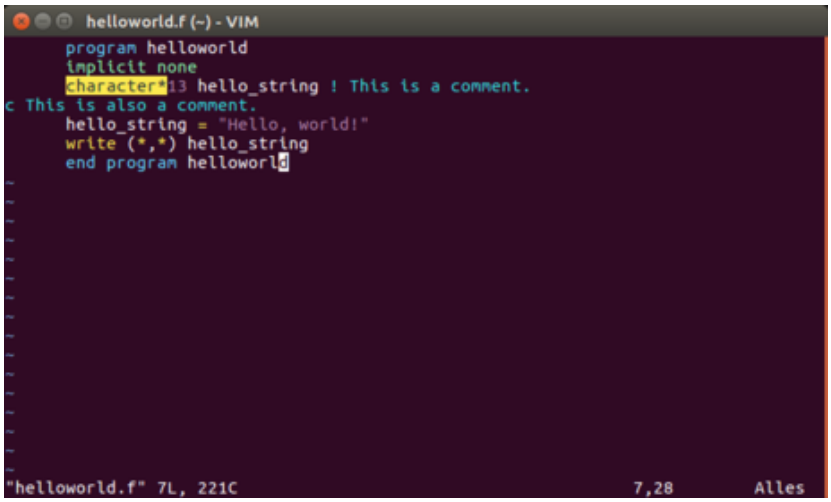
   Tip: The spaces are only necessary in versions of Fortran up to FORTRAN 77. If you're using a newer version, you can drop the spaces. Compile programs from newer version with f95, not f77; use .f95 as the file extension instead of just .f.

2. **Compile the program.** To do this, type `f77 helloworld.f` into the command line. If this gives an error, you probably haven't installed a Fortran compiler like for example gfortran yet.

3. 

Run your program. The compiler has produced a file called `a.out`. Run this file by typing `./a.out`.

4. **Understand what you just wrote.**
    1. `program helloworld` indicates the start of the program "helloworld". Similarly, `end program helloworld` indicates its end.
    2. By default, if you don't declare a variable type, Fortran treats a variable with a name that begins with a letter from `i` to `n` as integer, and all others as a real number. It is recommended to use `implicit none` if you don't need that behaviour.
    3. `character*13 hello_string` declares an array of characters which is called `hello_string`.
    4. `hello_string = "Hello, world!"` assigns the value "Hello, world!" to the declared array. Unlike in other languages like C, this can't be done on the same line as declaring the array.
    5. `write (*,*) hello_string` prints the value of hello_string to the standard output. The first * means to write to standard output, as opposed to some file. The second * means not to use any special formatting.

5. 

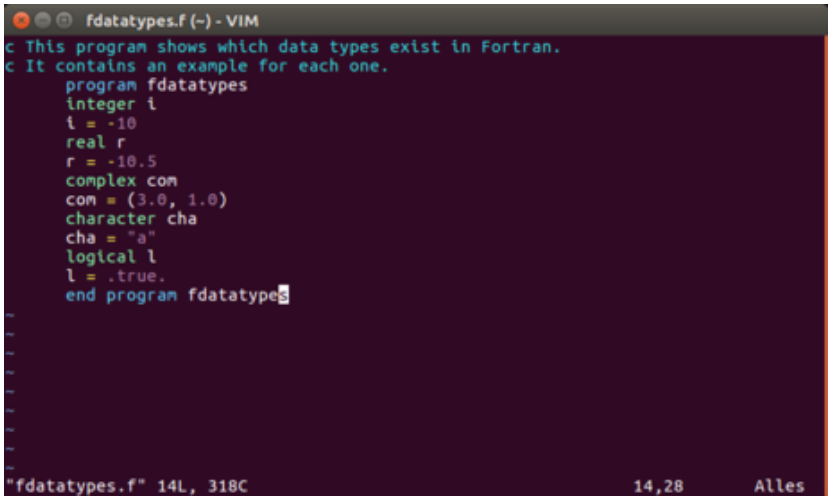
Add a comment. This isn't necessary in such a simple program, but it will be useful when you write something more complex, so you should know how to add them. There are two ways to add a comment.

1. To add a comment that has an entire line on its own, write a `c` directly into a new line, without the 6 spaces. After that, write your comment. You should leave a space between the c and your

comment for better readability, but this isn't required. Note that you have to use a `!` instead of a `c` in Fortran 95 and newer.
2. To add a comment in the same line as code, add a `!` where you want your comment to begin. Again, a space isn't required, but improves readability.

Part 2 of 4:

Using Input and If-Constructions

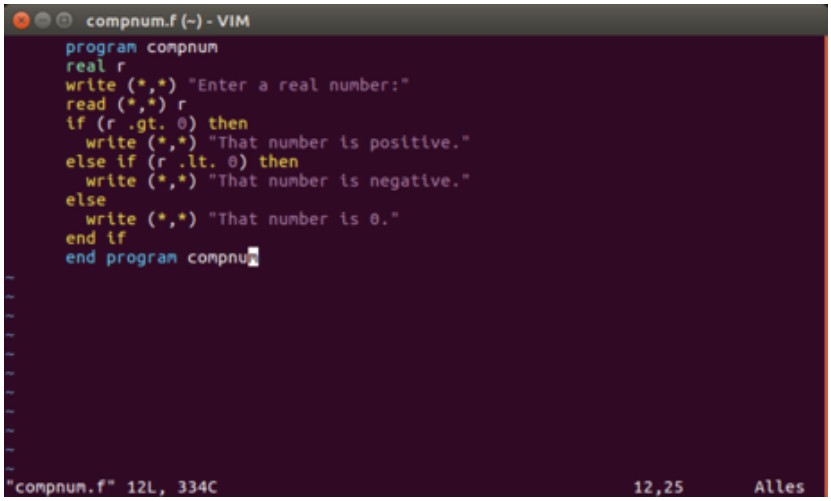
1. 

Understand different data types.

1. INTEGER is used for whole numbers, like 1, 3, or -3.
2. REAL can also contain a number that isn't whole, like 2.5.
3. COMPLEX is used to store complex numbers. The first number is the real and the second the imaginary part.
4. CHARACTER is used for characters, like letters or punctuation.
5. LOGICAL can be either `.true.` or `.false.`. This is like the boolean type in other programming languages.

2. **Get the user's input.** In the "Hello world" program that you wrote before, getting user input would be useless. So open a new file and name it `compnum.f`. When you've finished it, it will tell the user whether the number they entered is positive, negative or equal to zero.
   1. Enter the lines `program compnum` and `end program compnum`.
   2. Then, declare a variable of the type REAL. Make sure that your declaration is between the beginning and the end of the program.
   3. Explain the user what they're supposed to do. Write some text with the write function.
   4. Read the user's input into the variable you declared with the read function.

```
program compnum real r write (*,*) "Enter a real number:" read (*,*) r
end program
```

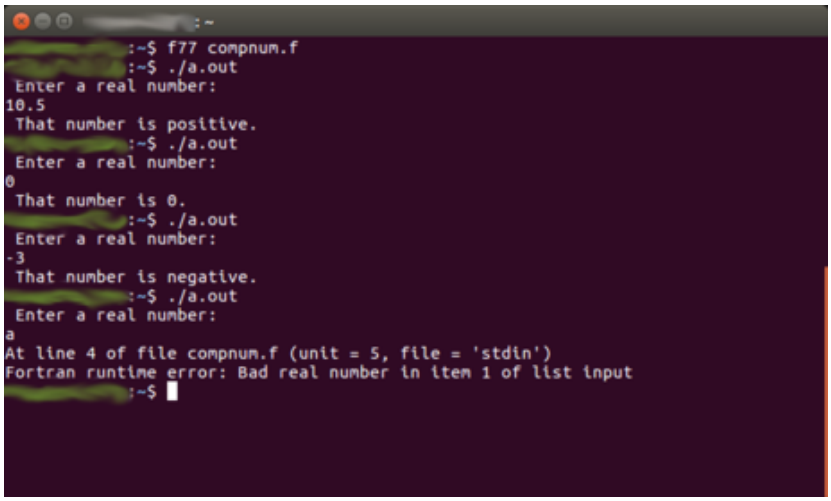
3. 

Process the user's input with an if-construction. Put it between the `read (*,*) r` and the `end program`.

1. Comparison is done with `.gt.` (greater than), `.lt.` (less than) and `.eq.` (equals) in Fortran.
2. Fortran supports `if`, `else if`, and `else`
3. A Fortran if-construction always ends with `end if`.

```
 if (r .gt. 0) then  write (*,*) "That number is positive." else if (r .lt. 0) then  write (*,*) "That number is negative." else  write (*,*) "That number is 0." end if
```

Tip: You don't have to indent code inside of if-constructions with more spaces, but it improves readability.

4. 

Compile and run your program. Input some numbers to test it. If you enter a letter, it will raise an error, but that's okay because the program doesn't check whether the input is a letter, a number, or something else.

Using Loops and Arrays

1. **Open a new file.** Since this concept is different, you'll have to write a new program again. Name the file `addmany.f`. Insert the corresponding `program` and `end program` statements, as well as an `implicit none`. When you're finished, this program will read 10 numbers and print their sum.
2. **Declare an array of length 10.** This is where you will store the numbers. Since you probably want a sum of real numbers, you should declare the array as real. You declare such an array with

   ```
   real numbers(50)
   ```

   (numbers is the name of the array, not an expression).
3. **Declare some variables.** Declare `numSum` as a real number. You will use it to store the sum later, but since `sum` is already taken by a Fortran expression, you have to use a name like numSum. Set it to 0. Declare `i` as an integer and don't assign it any value yet. That will be done in the do-loop.
4. **Create a do-loop.** The equivalent of that in other programming languages would be a for-loop.
   1. A do-loop always starts with `do`.
   2. On the same line as the do, separated from it by a space, is the label to which the program will go when it's finished. For now, just write a `1`, you'll set the label later.
   3. After that, again only separated by a space, type

      ```
      i = 1,10
      ```

      . This will make the variable `i`, which you had declared before the loop, go from 1 to 10 in steps of 1. The steps aren't mentioned in this expression, so Fortran uses the default value of 1. You could also have written

      ```
      i = 1,10,1
      ```

   4. Put some code inside the loop (indent with spaces for better readability). For this program, you should increase the variable `numSum` with the i-th element of the array `numbers`. This is done with the expression

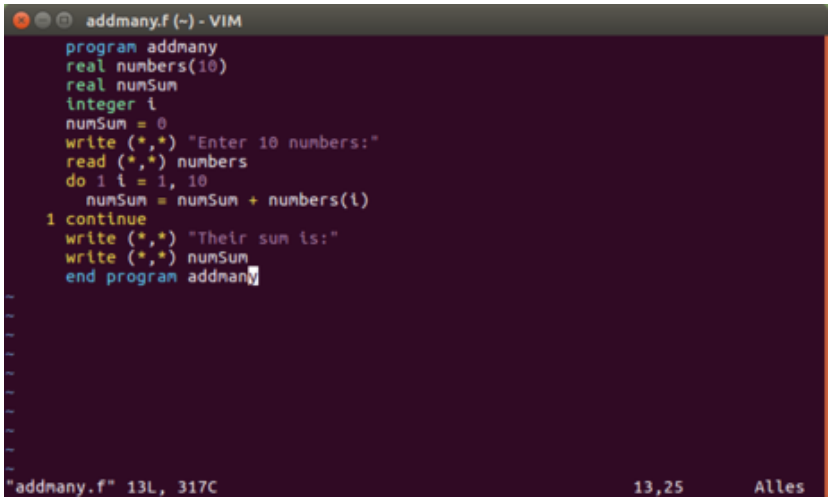
      ```
      numSum = numSum + number(i)
      ```

   5. End the loop with a continue statement that has a label. Type *only 4 spaces*. After that, type a `1`. That's the label which you told the do-loop to go to after it finishes. Then, type a space and `continue`. The `continue` expression does nothing, but it gives a good spot to place a label, as well as showing that the do-loop ended.

   Your do loop should now look like this:

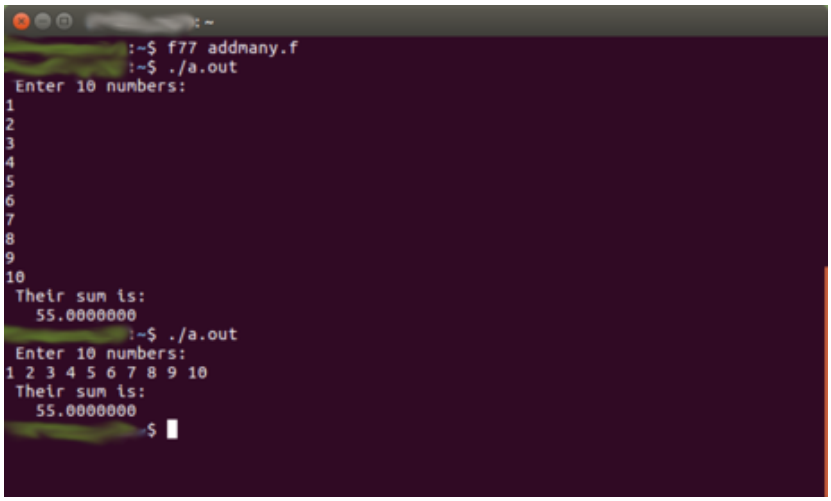
   ```
   do 1 i = 1, 10 numSum = numSum + numbers(i) 1 continue
   ```

   Tip: In Fortran 95 and newer, you don't need to use labels. Just don't put one into the do statement and end the loop with "end do" instead of "continue".

5. 

Print numSum. Also, it would make sense to give some context, for example "The sum of your numbers is:". Use the write function for both. Your entire code should now look as follows:

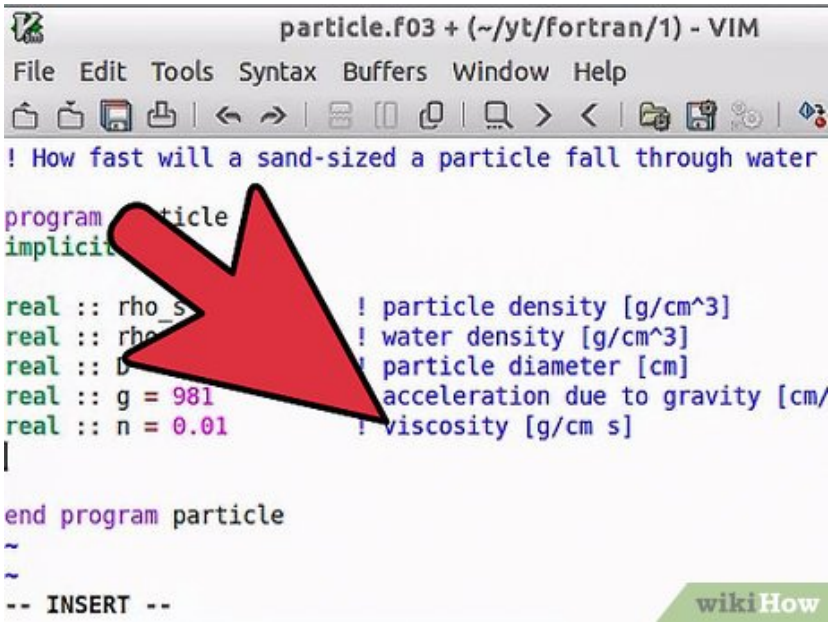
```
 program addmany implicit none real numbers(10) real numSum integer i
numSum = 0 write (*,*) "Enter 10 numbers:" read (*,*) numbers do 1 i =
1, 10 numSum = numSum + numbers(i) 1 continue  write (*,*)
"Their sum is:" write (*,*) numSum end program addmany
```

6. 

Compile and run your code. Don't forget to test it. You can either press ? Enter after each number you enter or enter many numbers on the same line and separate them with a space.

Part 4 of 4:

Understanding Advanced Concepts

1. 

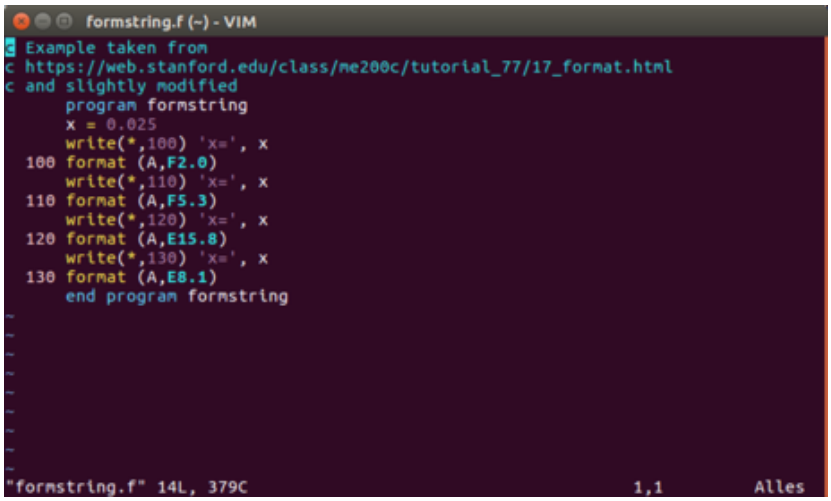
Have a good idea of what your program will do. Think about what sort of data is needed as input, how to structure the output, and include some intermediate output so you can monitor the progress of your calculation. This will be very useful if you know your calculation will run for a long time or involves multiple complicated steps.

2. 

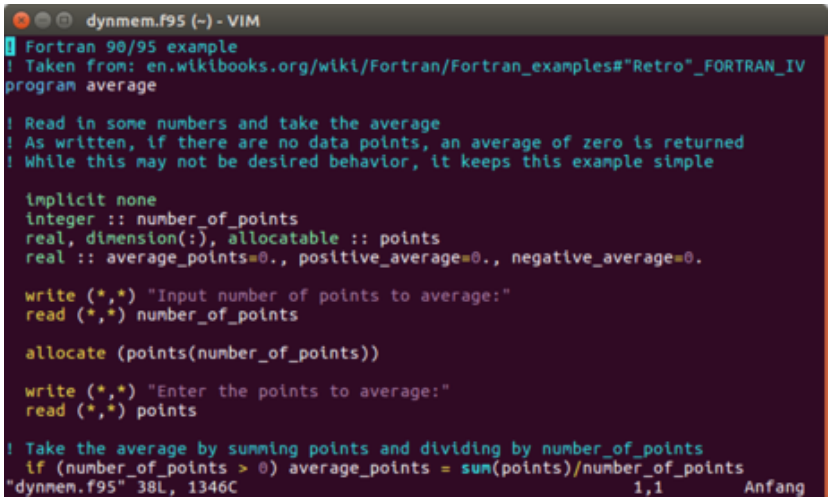
Find a good Fortran reference. Fortran has many more functions than explained in this article, and they might be useful for the program you want to write. A reference lists all functions a programming language has. This is one for Fortran 77 and this is one for Fortran 90/95.

3. 

Learn about subroutines and functions.

4. 

Learn how to read and write from/to files. Also learn how to format your input/output.

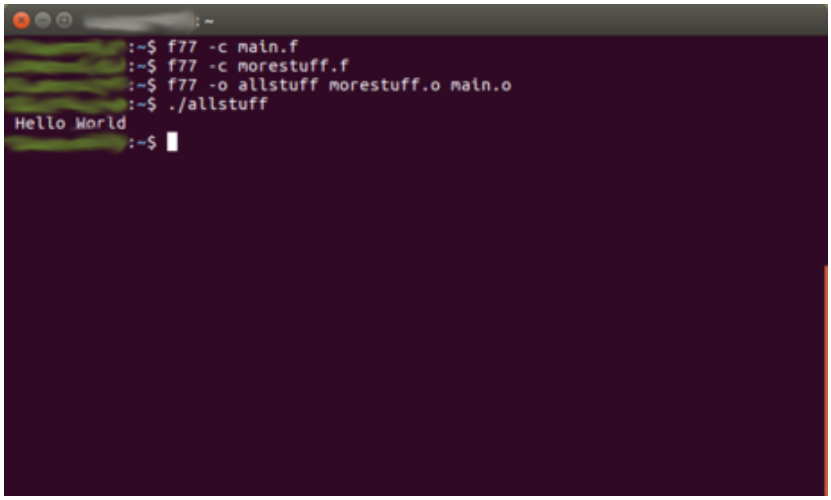
5. 

Learn about the new features of Fortran 90/95 and newer. Skip this step if you know that you'll only be writing/maintaining Fortran 77 code.

1. Remember that Fortran 90 introduced the "Free Form" source code, allowing code to be written without the spaces and without the 72 character limit.

6. 

Read or look up some books on Scientific Programming. For example, the book "Numerical Recipes in Fortran" is both a good text on scientific programming algorithms and a good introduction to how to put together codes. More recent editions include chapters on how to program in a mixed-language environment and parallel programming. Another example is "Modern Fortran in Practice" written by Arjen Markus. The book gives an insight into how to write Fortran programs in twenty-first-century style in accordance with the latest Fortran standards.

7. 

Learn how to compile a program spread across multiple files. Let's assume that your Fortran program is spread across the files `main.f` and `morestuff.f`, and that you want the resulting binary to be named `allstuff`. Then you'll have to write following commands into the command line:

```
f77 -c morestuff.f f77 -c main.f f77 -c morestuff.f f77 -o allstuff main.o m
```

Then run the file by typing `./allstuff`.

Tip: This works the same way with newer versions of Fortran. Just replace .f with the correct extension and f77 with the correct compiler version.

8. **Use the optimization your compiler provides.** Most compilers include optimization algorithms that improve the efficiency of your code. These are typically turned on by including a -O , -O2, or -O3 flag when compiling (again depending upon your version of fortran).
    1. Generally, the lowest level -O or -O2 level is best. Be aware that using the more aggressive optimization option can introduce errors in complex codes and may even slow things down! **Test your code.**