John Carter carte668
Csci 3081
Writing Assignment 4

How to Relieve Stress: Create a Makefile.

College students are a stressed group of individuals. Computer Science students are probably more stressed than most. I think most Computer Science students would agree that the most fun we have is doing projects where we can create programs, but that may be the most stressful part too. They take a lot of time and effort, and the worry that every time we start a project we may not get it done is often there. One of the greatest stress relievers I have encountered during the course of my schooling are Makefiles. Not only do they reduce the time it takes to create a program but they alleviate some of the stress of an already stressful process. Whether you have yet to utilize the beauty of a Makefile or have used it for multiple projects, understanding why a Makefile is such an important part of any project is the first step to reducing stress. I will discuss 2 primary advantages of using a Makefile followed by a quick discussion of the disadvantage(it’s quick because there really isn’t one). What follows is a brief explanation about how Makefiles work and why they are so important, written by a college student for college students.

Regardless of whether you have utilized Makefiles or not, they are rarely a required or graded aspect of a project. You are often told to create a Makefile and then get to programming. Too often this can lead to a student finding a few online tutorials and creating a Makefile by essentially copying an example, without really gaining a true understanding of what they are doing or even thinking about the advantages. This is not an exhaustive tutorial on all that you can do with a Makefile but rather is meant to show how to create a simple Makefile and the immense benefits you will receive. What follows are 2 primary advantages of utilizing Makefiles and once these ideas are grasped, you are on your way to much needed stress reduction. I will be using C++ for the examples of how to create Makefiles but this can be generalized to virtually any language, and in reality much more once you understand how Makefiles work.

Advantage 1: *Makefiles are time-savers*

For starters, let’s pretend we have a simple program “noStress.cc” that, when we execute it, says “Makefiles are great”. We will assume that you understand how to compile programs on the University of Minnesota lab computers (otherwise you will need to do some googling about compiling with GNU). To compile this program and execute it we simply need to type in the command line:
g++ noStress.cc –o noStress, which creates a file that we can then execute by typing: **./ noStress**, which will then print “Makefiles are great”. Seems simple enough, maybe not worthy of creating a Makefile for it, but I would like to show you why I believe it is.

The first advantage to creating a Makefile, even for a simple program like this, is the time saved by minimizing what is typed in the command line. You may be thinking that there isn’t much to type but let’s see why it is important. To state the obvious, typing: **g++ noStress.cc –o noStress**, then: **./noStress** takes a little longer than typing: **make**, and the word make is less prone to typing errors. And when you create a Makefile for this program, the word **make** is all you would have to type in the command line. That is much simpler. If you are still not convinced consider a program that needs 20 files to execute. Depending on how many header and source files there are you will either be typing some very long lines or a lot of short ones in the command line. If you haven’t yet created a program that has that many files then you will just have to trust that it would be rather time consuming to type all of that in the command line, not even taking into account typing mistakes, which will be numerous. Now consider that you will be modifying these files constantly until your program is finished, and each time you will have to retype some or all of these commands. If you properly create a Makefile for this 20 file program then you only have to type these commands once. Once you have written them in the Makefile all you have to type is **make** in the command line and magically your program runs. Except it isn’t magic, once you understand that a Makefile is just a substitute for what you would have to type anyway in the command line, you will realize that typing those commands once into a file is a lot faster, in the long run, than typing them every time in the command line. Not only is that faster but, when working on a stressful project, it helps to alleviate some of the stress you may otherwise feel. Which leads to the next advantage.

Advantage 2: *Makefiles are wonderful stress-reducers*

Let's consider the same example from before where we write and then execute the simple program that produces, "Makefiles are great". Granted, there is no real stress involved in creating that program, unless there was a 3 minute time restriction. But as any Computer Science student understands, when you are creating a program with multiple files, and you may not initially have much understanding of how to even begin, Makefiles truly are your best friend. Consider a program with 5 files, of which 4 are given to you, which is often the case in school. You need to write something in the 5th file that makes this program do something, and do it right. So you write a little code, enough to compile and execute the program. Except that it doesn't compile. You have numerous errors which you have to go in and, one by one, fix. Every time you do this you have to type 1 or 2 lines into the command line to make it compile and execute. Generally, you have to do this a lot. You've been working long and hard at this, making slow headway, and each time you make a change and have to re-compile your patience grows thin. You are prone to making more typing errors in the command line, and with each error you are becoming more and more aggravated. You finally reach a point where you realize half of the stress of trying to figure out how to create a program can simply be in compiling and executing it. This is where our Makefile comes in. Fix something in your file, type **make**, and repeat. Suddenly your stress reduces and you finally feel like you can start putting your efforts where they belong, trying to figure out this program. Now pretend you are writing a program with 20 files or more, now Makefiles become even more important. I don't even want to think about all of the aggravation I would feel if I had to type long and numerous commands every time I made a change and had to compile and execute. This 2nd advantage of utilizing Makefiles is by far my favorite, because it truly does reduce the amount of stress that goes into any project.

Demonstration: *it really is that easy*

Now that we have discussed the 2 main advantages of using Makefiles let's do a quick demonstration to see where we are at and show how truly easy it can be to create a simple Makefile. Let's take the example of the 5 file program that we are trying to finish. To compile and execute this program using some arbitrary file names, we would type something like this:

g++ main.cc file1.cc file2.cc file3.cc file4.cc –o programName, followed by: **./programName**. (You can imagine a program with a lot more files than this, including header files, all lying in different directories, which becomes significantly more difficult to type). Creating a simple Makefile to do this exact same thing is only slightly more difficult than typing those commands in the command line. First, create a file called Makefile or makefile which are the file names that the compiler recognizes as actually being Makefiles. Then simply type into that file:

```
all:  programName
   ./programName
programName:
   g++ main.cc  file1.cc  file2.cc  file3.cc  file4.cc  –o  programName
clean:
   rm programName *.o
```

You will agree that this is only slightly more difficult than using the command line, except you only have to type all of that once. The words **all** and **programName** in the above file are called targets, because that is what the Makefile looks for when wanting something to do, which is execute your program. They are followed by a colon which is how the Makefile knows they are targets. When you type **make** in the command line the Makefile either finds a target called **all** or, if no **all** is present, then the first target it sees starting from the top of the file. Immediately to the right of **all:** is **programName**, which you realize is the name of your executable that you would have run with **./programName**. The reason we put that there is so that when **all:** goes to the next line to run **./programName**, it realizes that in order to run **./programName** it first has to create **programName.** So the **programName** next to **all:** is called a dependency because **all** depends on **programName** first being created.

So the Makefile sees **all** and realizes it first needs to create the target/executable that you want created, **programName**. Before going further it jumps down to that target, **programName:**, which you'll notice in this simple program doesn't require, or isn't dependent on anything else. Because it doesn't see any dependencies it jumps to the next line **g++ main.cc file1.cc file2.cc file3.cc file4.cc –o programName** and runs that just like you would have done yourself in the command line. Once it runs that and creates an executable file, **all** can now do what it wanted to do initially which is run **./programName**, which is what you would have typed in the command line to execute your program. The last target, **clean**, is there to do exactly what it sounds like, to clean or remove any files that you specify, which are generally the executable or object files that you created to run your program. It's as simple as that. Now every time you make changes to your program and want to execute it you only have to type **make** in the command line. If you want to remove all of the files that **make** created just type **make clean** and, poof, they are gone (you, of course, have to specify which files you want "cleaned"). You could also have, instead of typing just **make**, typed
make programName, followed by **make all**, which would have first gone to the **programName** target, created the executable for it, and then when you type **make all**, **all** already has the **programName** executable that it needed so it can immediately run **./programName**. Even for a simple program with 5 files like this, hopefully you are starting to see the benefit of creating a Makefile for your program. It will, I promise you, make your life much easier and let you put your focus solely where it should be, on you project.

Disadvantages: *creating good Makefiles is a little more difficult*

For college students like us, there really aren't any disadvantages to using Makefiles. What we have discussed up to this point is how to create a simple Makefile, enough to reap some really good benefits. What we haven't discussed is how to make a really good Makefile, which is its primary disadvantage because, like anything new and worthwhile, it takes some effort to learn. What I hope is that I have instilled enough curiosity that those who have read this will seek out more information about Makefiles, which is aplenty. Let’s put it this way, if you invested $10 and I said I would get you a return of $10,000, you would take that in a heartbeat. It’s the same return you will get if you invest 5 hours learning about Makefiles, because I can guarantee a 5 hour investment on your part will return thousands of hours of programming benefits.