COP 1000 Introduction to Programming
Overview and Frequently Asked Questions (FAQ)

Jerry Reed

What will I learn in this class?

This is the first class for students who want, or think they want, to pursue computer programming. As such, we concentrate on teaching a few, broad concepts and techniques that can be applied to solving almost any programming problem in almost any language.

Here is the catalog description of the course:

> COP 1000. INTRODUCTION TO PROGRAMMING CONCEPTS.
>
> A hands-on introduction to analyzing, designing, coding, and testing computer programs. Students will develop algorithms for problem solving with an emphasis on good programming practices. Students will use programming techniques including control structures, files management, arrays, and subprograms to design and code basic programs using a modern computer language. Other topics include working with data, number systems, and an introduction to object-oriented and event-driven programming. This course prepares students for software development courses in programming and web development. Students with a demonstrated background in computer programming (transcript, job experience, or waiver exam) may request to have this course waived as a prerequisite to subsequent courses through the department office.

More detailed Learning Outcomes, that is skills you should have by the time you complete this class, are listed here:

http://fd.valenciacollege.edu/file/greed9/cop1000_learning_outcomes.pdf

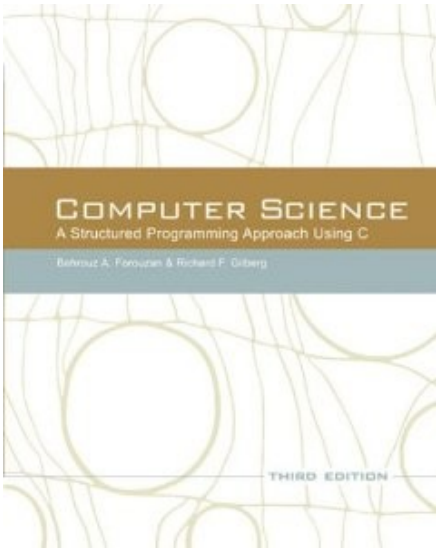
I try to introduce an element of challenge and interest to the material, while still staying on a basic enough level that everyone can succeed in learning the bulk of the material.

What textbook are we using?

Computer Science – Structured Programming using C (3rd Edition), Behrouz A. Forouzan and Richard F. Gilberg, ISBN 0534491324.

Available at the Osceola Campus bookstore, or via Amazon and others:

http://www.amazon.com/Computer-Science-Structured-Programming-Approach/dp/0534491324



Why learn C? Why not Java or *[insert your favorite programming language here]?*

An answer that probably won't seem too plausible for you at first is

> "in the long run, choice of your first language doesn't really matter all that much, because you'll end up learning multiple languages and they are really more similar than they are different".

Now since its a fair amount of work to learn a language, especially the first one, you may not really believe this statement now. And some programming languages say Python, look a lot different from C, while others, such as Java bear a marked similarity. However, and if you continue in programming, I think you will find you that, on the whole, the above statement is true.

That said, there are some features of C that still recommend it as a language. Although the C language has been around for quite a while, it remains quite popular for much application development. For some graphs related to this, please see:

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

As this page indicates, C is the number one or number two most common language turned up by Internet search engines. But even this doesn't tell the whole story of why C is important. As of June, 2013, Of the top ten most popular languages shown on this page, eight are based on or have syntax fundamentally similar to, C.

For instance, Java, Objective C, C++, PHP, JavaScript, Perl and C# all:

- use braces ("{"and "}") to group statements,
- have similar control structures (while and if) and
- feature similar operators (++, --, +=, ==, etc.).

So learning C serves as an excellent background for many other popular languages.

Where is C used today?

While C has been eclipsed by Java and PHP in many large-scale web applications, it remains the best available alternative in a couple of domains.

Where system-level software is needed.

All Unix-based operating systems, and this includes Linux and Mac OS X, are written primarily in C, with some C++. C works well for operating systems because it is efficient of machine resources, executes quickly, and is reasonably portable among differing hardware platforms.

Where performance is king. ("I feel the need...the need for speed!")

C is still widely used where performance, that is some measure of throughput, must be optimized. This might be megabytes transferred on a LAN to a storage medium, or frames per second, number of concurrent sessions, or any measure of speed and performance one can imagine. Because it is the "low-level high-level" language, C compilers can generate some of the most efficient machine code possible short of actually writing in assembly language. This contributes to a well-earned reputation for good performance.

Where resources are scarce.

Some of the very things that make C a more difficult language for beginners than say Visual Basic are the very features that allow C to run in environments with limited memory and limited processor speed:

- Pointers, with expressions like (*p) = I ;, *(we won't get this far)*
- low-level, non-automated control over memory allocation (malloc and free), *(wait for COP 2220)* and
- reflection of the underlying hardware in the operators and syntax of the language (e.g. ++ for increment)

all contribute to C programs being compact enough and fast enough to function on machines with only a few kilobytes of memory, or clock rates in the 10's of megahertz.

Where direct communication with computer hardware is required.

Programs such as:

- device drivers that allow your programs to operate disks, USB devices, video displays, and the like,
- the whole range of embedded applications, those computer systems embedded in cars, DVD players, cell phones, toys and more, and
- various Physical Computing devices, that help software interact with the real world,

all benefit from C's low-level features and efficiency.

So when you're learning C, you're learning a language that is:

- Still immensely popular,
- A stepping stone to many other popular languages, and

Useful for entertaining projects that connect inexpensive hardware to the real world.

This last reason is the one we're going to explore in our C class, as we apply C to Physical Computing problems. Follow the links on this page to read more about how the class will work.

What tools are we using?

We'll be using CodeBlocks, a C/C++ IDE.

http://www.codeblocks.org/

We'll also use some pre-written components for things like graphics and sound from OpenFrameworks:

http://www.openframeworks.cc/

Most programmers today, regardless of programming language, use an *IDE* – an Integrated Development Environment. By integrated, we mean that the tool combines an editor, a compiler and linker, a debugger, and other components into a single program.

IDEs of which you may have heard include Eclipse, Visual Studio, NetBeans and CodeBlocks. We'll be using CodeBlocks, but more on that in a moment.

The big advantages of an IDE are that all the tools you need are in one place, and they've been customized to work together. Once you finish editing your program by typing in the programming language, you can compile it, translating it from human-readable to machine-readable, with a single click. Similarly, once the program compiles successfully, you can click another button to start a debugger, which lets you step through the operation of your program, one line at a time to find and fix mistakes.

The main disadvantages are that, like any complex application program (say Excel), IDEs have many features and a significant learning curve. This means that it takes some effort and time to become familiar with how to use the tool to write and run your programs. This is on top of learning the language itself.

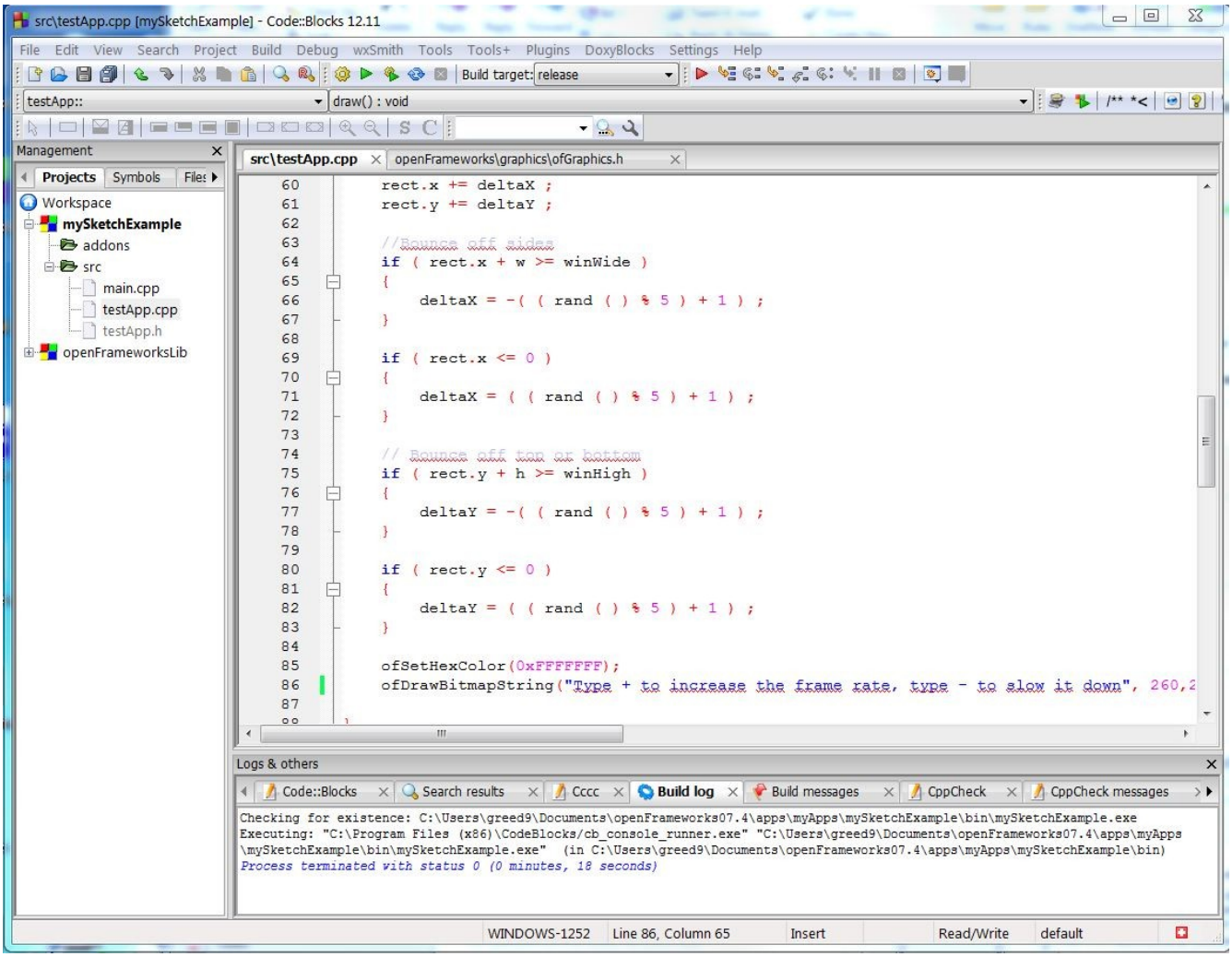
The other disadvantage of an IDE is that you will have to install it on your PC or Mac, and while that usually goes off without a hitch, on some computers, for some reason, there will be challenges just getting the tool to run. We are prepared to help with that, too. More on this under resources below.

However, once you get familiar with the tool, you'll see how the program features (syntax highlighting, debugging, auto-completion, context-sensitive help, auto-formatting, etc., etc.) can help you write better programs more quickly than you could with say, Notepad or TextEdit.

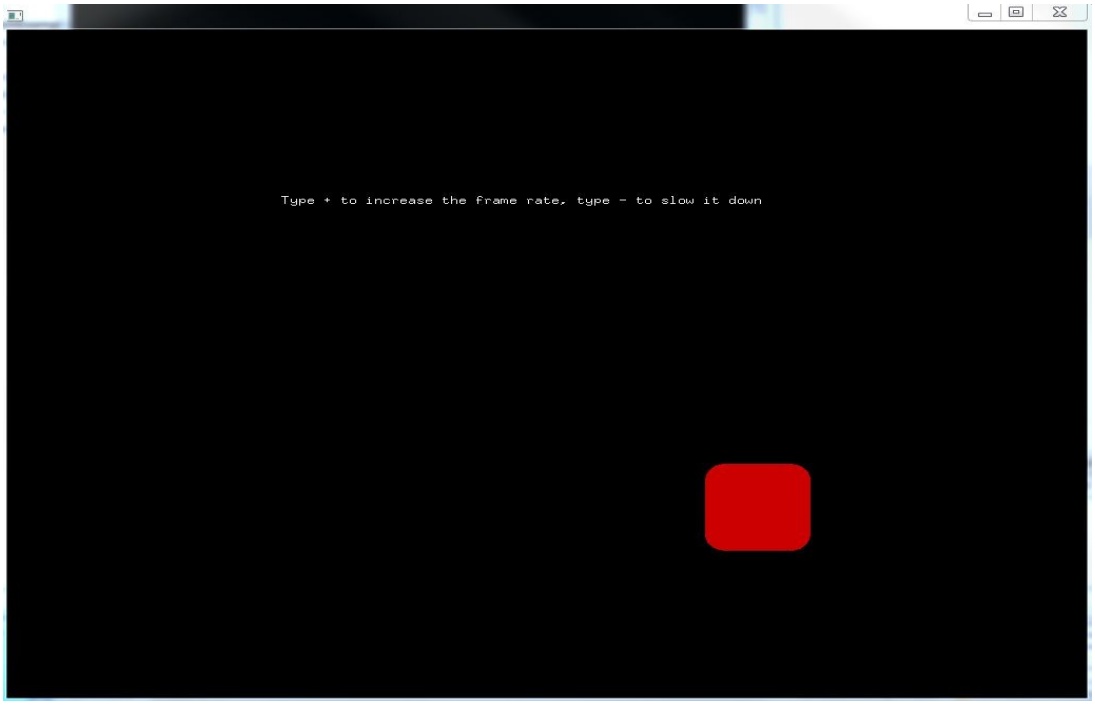
As I said, we'll be using CodeBlocks.

http://www.codeblocks.org/

Here's a screenshot to give you a feel for what it looks like:



The CodeBlocks screen above shows editing the source code for the demo program whose output looks sort of like this:



Not very exciting as a static screenshot, huh? When you run it, the rounded rectangle changes color and bounces around the screen, speeding up or slowing down as you press the "+" and "-" keys.

There's a short video clip of the program running on YouTube:

http://youtu.be/JNdj-PVrAJ8

The CodeBlocks project/workspace containing the source code is on this page, too:

http://frontdoor.valenciacollege.edu/materials.cfm?uid=GREED9

As you can see, we'll also be using OpenFrameworks, a collection of open-source software that supports all sorts of interesting program behavior, such as graphics, sound, hardware interfacing, and so on. The demo program above was built with OpenFrameworks.

What will the class be like?

The syllabus will spell this all out in greater detail, but for now, it may help to understand that our class will be composed of several different types of activities.

1. Reading the assigned chapters and/or related web links. Read my notes on each chapter along the way.
2. Quizzes on the assigned chapters and web links. These will probably be weekly.
3. A midterm exam and a final exam.
4. Projects that ask you to write or complete programs. These will range from easy to challenging.
5. Discussion Postings and replies – from time to time I'll ask you to post your project code, or

output, or web links you used in the Blackboard Discussion forums to share. I also will give credit for help you supply to questions from other students (and for really good questions you ask) in the Questions and Answers Discussion forum.

Let's look at examples of these, one-by-one:

1. We'll probably cover the following eight chapters in your text:

- Chapter 2, Introduction to the C language
- Chapter 3, Structure of a C program
- Chapter 4, Functions
- Chapter 5, Selection – Making decisions
- Chapter 6, Repetition
- Chapter 7, Text input and output
- Chapter 11, Strings
- Chapter 12, Enumerated, Structure and Union types.

Some chapters we'll spend more time on that others, and of course, in order to write or complete the program for an assigned project, you may need to skip around in a bit in the book or look at other chapters to figure out how to solve the problem.

The textbook is an incredibly through explanation of C, with a wealth of detail that is relatively unimportant in an introductory class. I'll supply notes for each chapter explaining my take on the material, and indicating what I think is most important, what can be skipped for now, and probably where and why I disagree with the authors occasionally.

2. Here are a couple of sample Quiz items.

From the web/project work:

> Briefly describe one important difference between a Workspace and a Project in CodeBlocks.

How would you edit and compile a file with CodeBlocks without first creating a project?

From Chapter 2:

> Which of the following is a valid identifier in C?
>
> a. a3
> b. 4A
> c. a-b
> d. if

From Chapter 3:

> In C, a compound statement is contained one or more lines between which two characters?
>
> a. (and)
> b. [and]
> c. { and }
> d. “ and “

From Chapter 11:

> In C, a string is stored as an _____ of characters.
>
> a. structure
> b. array
> c. list
> d. index

3. The midterm exam will consist mainly but not exclusively, of items from the weekly quizzes, and is likely to have an added “write some code” section. The final exam will be comprehensive.

4. I'll have a range of projects, varying in complexity. Your goal will be to accumulate at least a certain minimum of points, either by doing a greater number of smaller and easier projects, or a lesser number of more challenging and demanding projects.

An example of an early and fairly easy project might be:

> **Better bounce**
>
> ...add code to a “template” program in some creative fashion to alter the behavior of the “bouncing square” demo described above. (Change speed, change shape, play a sound, etc. etc.)

There will be a few projects that don't directly involve writing code thrown in, too. Here's an example of one of those:

> **C-like languages on the web**
>
> “...you are asked to investigate a language that is based on, or at least similar to, C. Your task is to compare and contrast the language you select from the list with C...
>
> Answer the following questions for each language you select:
>
> a) How are variables implemented?
>
> b) Are there more or fewer data types than C?
>
> c) How is conditional execution (if/else) implemented? Specifically, how does this differ from C, if it does differ?
>
> d) How are loops written? Does the language have any built-in functionality that makes explicit for loops less necessary?
>
> e) Name and briefly describe at least one feature of the language that is not found in basic C.

f) Provide a short code sample in each of the two languages."

An example of a more challenging coding project is:

Simplified Image Processing

In this project you are asked to:

1. Define a two-dimensional array in your C program that can hold an image of up to 1024 by 1024 pixels, where each pixel should be represented as an integer.

2. Load the specified image from a file into the array in your program.

3. Perform some transformation on the input image to alter it systematically.

4.Write out the transformed image back to a file.

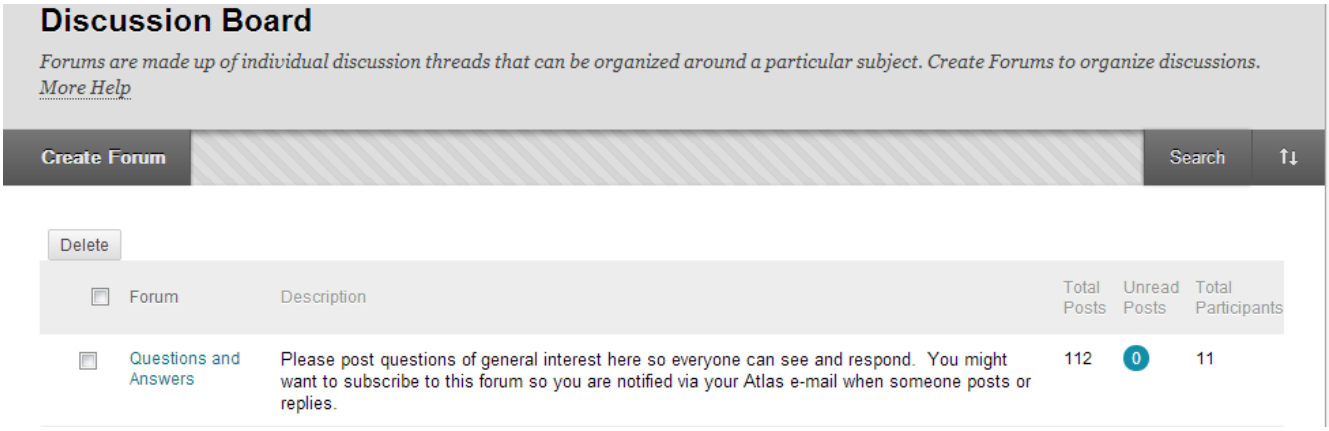
I'm providing C code for steps 2 and 4, so the required file operations (opening, reading, writing, closing) are there for you to see, but you need not code these yourself...

You need to add code for Step 1 (easy) and Step 3 to the program code I supply to form a complete image processing program. There are several variations for the processing you perform in Step 3, and they range from pretty straightforward to moderately challenging, with differing points available based on the amount of work required.

This is an on-line course. What resources are available if I need help?

You can ask a question in the Questions and Answers discussion forum (best), or e-mail me, or come to the West Campus IT lab.

Asking in the Questions and Answers Blackboard forum is the best and quickest option because:



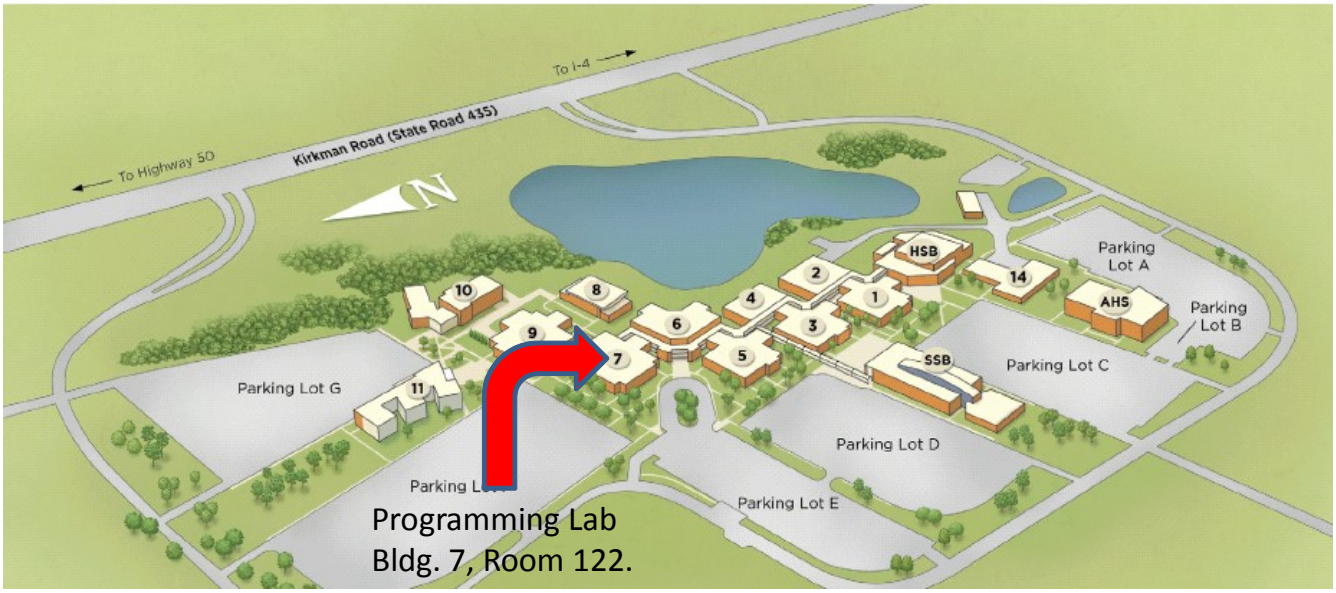
1. I subscribe to that forum, and you should too. That way as soon as something is posted, we'll get an e-mail with the post inside. Those go to my phone. I can usually respond to these within a couple of hours, between about 8:00 AM and 10:00 PM. *Please do not use the Messages facility in Blackboard (I try to hide it) because it does notify me and I will forget to check it in a timely fashion.*

2. Since there are almost certain to be a few folks in our class who already know a fair amount about programming, or who have already dealt with similar issues in their own work, I'm not the only one who may answer and provide help. Interaction with other folks on-line is a crucial part of this class.

3. You may find that your question has already been asked and answered already. If you see a question where you can contribute, please jump in and don't wait for me, or feel free to correct or extend answers from others and from me.

E-mailing me is fine, too, (greed9@valenciacollege.edu) but is best reserved for questions that you don't feel comfortable sharing more publicly, such as questions about your grades or progress in the class. I check pretty regularly, but not so much on weekends and nights, so the post with a push to my phone works much faster, in general.

Coming to the lab is logistically difficult, I know but for some problems, there is no substitute for hands-on assistance. The lab staff can help with the majority of shell scripting problems, and they can consult me if needed as well. This is particularly useful if you are having trouble getting your Linux system set up initially.



http://valenciacollege.edu/west/engineering/LabHours.cfm

If the Osceola campus is more convenient for you, the following information from the Lab Manager there may be helpful:

> When students need to work on lab assignments outside of class, they may go to the library in room 4-202. The web page denoting the open hours is below.
>
> http://valenciacollege.edu/library/hours.cfm
>
> Students may also use the Learning Center in room 3-100 to complete assignments outside of class time. The Learning Center open hours for the fall term have not been finalized yet but are generally Monday - Thursday: 8:00 am - 9:30 pm, Friday: 8:00 am - 5:00 pm, and Saturday: 8:00 am – 12:00 pm.
>
> We also have open lab hours in building three in rooms 3-301, 3-330, and 3-331 when classes are not in session. These hours very according to class scheduling. When the fall term class schedule is completed, I will be able to give you the open lab hours for these rooms.
>
> For tutoring information and scheduling, please contact Kimberly Foster at 407-582-4112 or email:

kfoster8@valenciacollege.edu.