# Personalize this Document by Viewing it in Word - Office 365:

* Click on the "View" tab above
* Click on the "Immersive Reader" button located near the top left
* Change the font, spacing, page contrast, use translator, listen to the document...

Introduction to CTEC1332

Mike Boldin will be giving lectures on program design using a simple pseudo-code for planning well-structured programs. Mike will also be supervising you in the lab. He is here to help you -- to make sure that you can get your programs to run.

This is one course, and you can talk to the instructor when you want to suggest an improvement. We know that any course can be improved and your suggestions for improvement will be received with thanks. Our goal is to make you better programmers; if you tell us how we can do that better, we will listen eagerly.

## The growing importance of software in Engineering

Programming is an important skill for all engineers and technologists. Software is growing in importance in every area of Engineering. For example, students will find that modern cars are loaded with on-board computers. Computers control the braking system, ignition system and automatic transmissions. Civil engineers are now designing computerized traffic control systems. Modern aircraft are controlled by a computer under the direction of the pilots. Power plants are controlled by computers. Most important, even products that do not have a computer in them are designed using computers.

## Most programs are incorrect

Errors, usually called "bugs" are commonplace in software. This statement is not about student programs but about professional programs; it is not about incomplete programs but programs that are released for use. Some well-known operating systems have thousands of known errors. Whenever a program, even a program that appears to be operating well, is subject to careful disciplined inspection, bugs are found. Sometimes the bugs are minor, but often they are serious; errors in programs have led to the loss of life and damage to property.

One of the most frustrating aspects of programming is that even tiny errors can have major effects. In programming, "almost right" is often a euphemism for wrong. Careful attention to detail, and the ability to deal with many arbitrary facts, is essential to programming.

Many studies have been made of the cause of program bugs. Here are some important observations:

* Most errors are interface errors, that is errors caused by a misunderstanding about how one program is to communicate with others.
* The second most frequent cause of errors is failure to consider *all* the cases that can arise when a program runs. In complicated programs programmers simply forget about situations that seem unlikely.
* Poor "structure" is a major cause of errors. Unless a program can be understood as a set of easily understood smaller component programs, programmers cannot understand them well enough to get them right. Unstructured programs can only be understood as a whole, not in small pieces. Unstructured programs have more errors that well-structured programs.
* Programming errors are caused by a failure to plan programs carefully; when programmers prepare program plans before they write the actual programming language code, they make fewer errors.

## A course on how to program well

The purpose of this course is to teach you how to program in a disciplined and systematic manner. You will be taught to use discipline and logic to reduce the number of errors in the programs that you produce. The lectures in this course are designed to teach you:

* The syntax and semantics of a practical programming language, C, while is extensively used in the communications industry and in such areas as robotics.
* How to use basic operating system facilities to produce and store programs and data.
* How to specify what a program does so that you can avoid interface errors.
* How to organize large programs as assemblies of smaller building-blocks so that a program can be understood, a little bit at a time, by those who must work with it.
* How to plan your program in a simple program-planning-language that helps you to get a well-structured program.
* How to translate those programs into C.

In other words, this course will teach you the discipline of programming - a procedure for planning, analyzing and implementing programs.

## The components of this course

### Lectures on how to design and analyze programs

You will learn the basics of programming in any language. You will be taught how to put programs together from smaller components and how to analyze programs to be sure that they are complete. In this part of the course, you will write programs developed in a pseudo-code designed to help you to plan a well-structured and easily analyzed program.

### Lectures on how to translate program plans into C

The pseudo-code used to develop programs is not a programming language. It is designed for people, not computers. To get your program to run, so that you can use it, you must translate your plans into a programming language that runs on the computer that you use. These lectures, and the textbook will help you to get started.

### Laboratory Assignments that must be done on a computer

You will be given programming assignments where you must plan a program, translate it into C, compile it, and test it thoroughly.

The components are integrated. Every program that you see developed in the lectures on program design, will be translated into C. Homework assignments are designed to help you learn how to use what you have heard in the lectures. All three components are vital; they are the three legs of a tripod.

## How to learn to program well

To learn how to program well you must write and run programs. This course provides you with a number of required assignments. As in any course, it is always a good idea to do extra assignments. The reference book contains other problems that you can try.

Your success in learning to be a good programmer depends as much on you as on us. Programming, like bicycle riding, cannot be learned by listening to lectures and reading books. The more you do it, the more you will learn. Even the "falls" are instructive. Only when you experience how picky the computer is, how even the smallest details matter, will you understand the need for the discipline that we are teaching in this course.

If you want to learn to program well, you should use the program design discipline that we teach. The problems in this course are designed for beginners. Many of you will find that you can take short-cuts, i.e., write the C without first planning the program. This works adequately for small programs, but it won't work well for the larger programs that you will have to write later on in your education and in your career. Only if you follow the discipline carefully, will you be able to write large programs that can be trusted.

Some of you have learned to use a computer, either in your previous schools, in the CTEC1184 course, in a job situation, or on your own. This means that you already know some of the mechanics of using a computer and the language syntax. While this is obviously an advantage, it is also a disadvantage. You have probably learned to write programs without going through the discipline of planning them in pseudo-code. You will have to change your habits -- if you want to get full value from this course.

### What we will teach

You will learn a programming language in this course, because practical exercises are essential to learning programming. We will discuss:

1. the basic nature of computers, how they can be described by state transition tables, and how state transition tables can be used to solve certain types of programming problems;
2. techniques for providing readable descriptions of programs;
3. how programs are constructed from primitive programs using basic programming constructs;
4. a simple notation for developing well-structured programs;
5. the development of programs that are well-structured and documented;
6. some interesting and useful algorithms.

### What you should learn

In this course you are going to be learning two quite different things at the same time. You will be learning details about a particular tool. Often, these details are arbitrary. In many cases the designers of the language would do things differently today. You learn those arbitrary facts because it is necessary to know how to use today's tools and because you cannot learn programming without running programs.

You will be learning how to think about programs, how to organize programs, how to describe what programs do. These principles will be useful next year, but they will be even more important when you have to develop large programs for your employer. It is important that you distinguish between properties of one language and programming fundamentals. Your ability to make that distinction will help you when you have to write programs in a language other than C.

## How to be a Good (Software) Engineer….



## References

[1] Neumann, P.G. "Computer Related Risks", ISBN 0-201-55895-X, 1995, ACM Press, Addison Wesley.

[2] Wiener, L. R. "Digital Woes, Why We Should Not Depend on Software", ISBN 0-201-62609-8, 1993, Addison Wesley.

[3] Parnas, D. L. "Engineering Computation", January 1995, McMaster University.