Miami Dade College

Course Description

COP2270 | "C" for Engineers | 4.00 credits

This course is intended for students majoring in Computer Engineering Technology, Electronics Engineering Technology, or any engineering discipline. Students will learn the C programming language, MATLAB, and the Engineering Problem Solving Method to analyze, design, code, compile and execute programs that solve engineering related problems. Pre/Corequisite: MAC1105. Recommended Preparation: CGS1060C or knowledge of computer skills.

Course Competencies:

Competency 1: The student will demonstrate an understanding of computer hardware fundamentals by:

1. Describing the architecture and operation of a typical computer system
2. Drawing a block diagram of a typical computer system, including input, output, CPU, storage, and main memory
3. Identifying the major storage devices, including registers, main memory, and secondary storage, and the advantages/disadvantages of each
4. Identifying the major areas of program data, including the stack, code segment, heap, and data segment

Competency 2: The student will demonstrate an understanding of computer software fundamentals by:

1. Drawing a block diagram of the compilation process, including input source code, compiler, assembler, and executable code
2. Describe the difference between static linking and dynamic linking libraries
3. Explain why a program compiled for one operating system may not run on a different operating system
4. Using and applying a "C" language build system to create executable programs from source for files, e.g., the gnu make build system
5. Compiling and linking a program in Posix and Microsoft Windows environments

Competency 3: The student will demonstrate the ability to compare programming languages and software by:

1. Identifying the differences between high-level and low-level languages
2. Discuss the differences between managed and unmanaged code
3. Describe the differences between interpreted and compiled languages
4. Identifying the advantages/disadvantages of common languages such as assembly, C, C++, Mat lab, java, c#, Perl, PHP, etc
5. Selecting the most suitable programming language given a set of application requirements.

Competency 4: The student will demonstrate mastery of algorithm development and flowcharting by:

1. Writing pseudo code for program development before writing code
2. Applying functional decomposition techniques to break a programming design problem into smaller pieces
3. Incorporating adequate and meaningful comments into the source code of programming projects
4. Participating in a team to develop a solution to a problem
5. Testing and debugging c program logic and code (using a debugging tool like the gnu debugger) by setting breaks and stepping in and out of code to examine memory contents at runtime
6. Explaining and implementing top-down design
7. Implementing the structured programming model
8. Evaluating alternative solutions and error conditions
9. Generating test data sets

Competency 5: The student will demonstrate an understanding of the internal representation of data, data types, and operators by:

1. Explaining the difference between the decimal and binary systems
2. Converting numbers into decimal, binary, and hexadecimal representation

3. Converting ASCII characters into their binary and hexadecimal equivalent
4. Defining floating-point numbers, their purpose, and why they involve more expensive operations
5. Describing the difference between little-endian and big-endian memory structure
6. Creating programs that use all available data types, including integer, character, floating point, double precision floating point, and void
7. Identifying the bit size and range of values of data types, including integer, floating point, double precision floating point, and void
8. Describing how the prefixes signed, unsigned, short, and long affect the bit size and range of the values of data types
9. Creating programs that use all the available operators, including addition (+), subtraction (-), multiplication (*), division (/), modulus division (%), equality (==), increment (++), and decrement (-)
10. Explaining the properties of a variable, such as its name, value, scope, persistence, and size

Competency 6: The student will demonstrate an understanding of control structures and data files by:

1. Creating programs that use if, else, if, and else statements to evaluate conditions
2. Creating programs that use logical operators such as and, or, and not in conditional statements
3. Creating programs that use nested conditional statements
4. Creating programs that use switch, case, and break conditional structure
5. Creating programs that use while, do-while, and for and nested loops to create repetition
6. Describe the conditions under which it is more practical to use a for loop than a while loop
7. Analyzing existing programs with loops and determining the results
8. Writing a program that reads an existing sequential file
9. Writing a program that creates and writes to a sequential file
10. Writing a program that produces formatted printed output

Competency 7: The student will demonstrate a mastery of c functions by:

1. Creating functions that use call-by-reference and call-by-value
2. Modifying existing programs that use functions
3. Creating programs with functions that return values
4. Identifying the scope of variables within functions
5. Creating programs with preprocessor directives such as #include and macros such as #define
6. Explaining recursion and identifying situations in which recursion should be used
7. Analyzing and creating programs that use recursive functions
8. Creating a program that takes advantage of recursive functions

Competency 8: The student will demonstrate an understanding of pointers, data structures, and file input/output by:

1. Describing the differences, advantages, and disadvantages of using a pointer variable over a regular variable
2. Analyzing programs that make use of pointers
3. Creating programs that initialize and use pointers and then destroy pointers
4. Creating a program that uses malloc and collects access run-time storage allocation
5. Explaining the form and uses of an array
6. Creating a program that uses single- and multi-dimensional arrays
7. Evaluating existing programs that search arrays
8. Defining the uses of structures
9. Writing a program that makes use of structures
10. Identifying the differences between advanced data structures such as stacks, queues, and linked lists

Competency 9: The student will demonstrate the ability to use c and the engineering problem-solving method to solve engineering problems by:

1. Describing the engineering problem-solving methodology
2. Defining an engineering/scientific problem statement

3. Describing the input and output information
4. Solving for the solution by hand for a simple data set
5. Outlining the various blocks/functions/steps required to solve the problem
6. Developing algorithms in pseudo code or by creating a flowchart to solve problems
7. Writing a c program that solves the problem
8. Testing the solution with a data set

Competency 10: The student will demonstrate an understanding of MATLAB by:

1. Performing basic mathematical computations using MATLAB
2. Using basic operators, variables, vectors, and arrays, basic and complex matrix operations
3. Creating graphical representations of data in MATLAB
4. Programming with m-file scripts that contain functions for loops, conditional statements, and while loops
5. Using MATLAB to prototype computing algorithms before implementation in C
6. using MATLAB to generate and visualize data and data files

Competency 11: The student will demonstrate an introductory level of understanding of the C++ language by:

1. Describing the object-oriented programming model and the standard template library
2. Analyzing a C++ program structure that solves an engineering problem
3. Analyzing a program that uses the input/output capabilities of C++, including reading and writing files
4. Analyzing several C++ programs that illustrate simple computations, loops, functions, 1 and 2-dimensional arrays, and data files
5. Describe how to define a class using class declarations and implementations
6. Analyzing programs that use new, delete, and constructors when defining classes
7. Describing inheritance, polymorphism, virtual functions, and abstract classes
8. Testing and debugging C++ program logic and code using a debugging tool, e.g., the gnu debugger, by setting breaks and stepping in and out of code to examine memory contents at runtime
9. Examining stack unwinding using catch/throw to process exceptions in C++ programs that solve an engineering problem

Learning Outcomes:

- Solve problems using critical and creative thinking and scientific reasoning
- Demonstrate knowledge of ethical thinking and its application to issues in society
- Use computer and emerging technologies effectively