C S 20A: PROGRAMMING IN C#

Foothill College Course Outline of Record

Heading	Value
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Advisory:	One of the following: C S 1A, 2A, 3A or equivalent.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to write and debug C# programs which make use of the fundamental control structures and method-building techniques common to all programming languages. Specifically, the student will use data types, input, output, iterative, conditional, and functional components of the language in his or her programs.
- A successful student will be able to use object-oriented programming techniques to design and implement a clear, well-structured Java program. Specifically, the student will use and design classes and objects in his or her programs.

Description

Introduction to the C# programming language and the .NET platform. Topics include object oriented programming, graphical user interfaces, elementary data structures, algorithms, recursion, data abstraction, code style, documentation, debugging techniques and testing.

Course Objectives

The student will be able to:
A. Describe the basic components of the C# software development environment.
B. Describe the C# software development life cycle from concept design through documentation, testing and maintenance.
C. Produce clearly written code in an industry standard style appropriate for C#.
D. Define both primitive and compound data types and give examples in C# of each type.
E. Write C# applications that define operators, use delegates, add event specifications, and implement predefined and custom attributes.
F. Incorporate user-interaction input and output in a program through both the console and a graphical user interface.
G. Define, analyze and code the basic C# conditional and iterative control structures and explain how they can be nested and how exceptions can be used.
H. Design, implement, test, and debug functions and methods that can be used in programs, and demonstrate the way parameters are passed in such functions and methods.
I. Write C# programs using object-oriented design, and contrast the difference between object-oriented and procedural code.
J. Write C# applications using derived classes and demonstrate correct use of inheritance.
K. Produce a program that interacts with the user using intermediate GUI elements, such as buttons and text-boxes.
L. Create analytical algorithms that use arrays for solving simple problems.
M. Explain methods of error handling and exceptions.
N. Explain what an algorithm is and give examples of how algorithms are implemented in a C# program.
O. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: string processing, numeric computation, simple I/O, arrays and the .NET framework collections.
P. Solve problems that have origins in a variety of disciplines, including math, science, the Internet and business.

Course Content

A. Introduction to C#
1. Structure of a C# program
2. The C# run-time environment
3. Compiled vs. interpreted languages
4. Comparison of major languages
5. Compiler errors vs. run-time errors
B. The Software Development Environment
1. The .NET framework
2. Toolbox
3. Properties window
4. Using the help system
5. Debugging and using the debugger
C. Coding Standards, Conventions and Styles
1. Acceptable indentation options
2. Naming conventions for variables and methods
3. Separation of computation and I/O
4. Modularity
D. C# Data Types
1. Value types and reference types
2. Arrays
3. C# compound types
4. User-defined data types
5. Namespaces
6. Boxing and unboxing
E. Operators and Events
1. Numeric operators and expressions
2. String operators and expressions
3. Logical operators and expressions
4. Overloading operators
5. Using and creating delegates
6. Using and defining events
F. Basic Input-Output Strategies
1. Console I/O
2. Simple GUI I/O
3. Formatting values for clean output
G. Control Structures
1. Selective and conditional statements
2. Loop statements
3. Nesting levels in control statements
4. Handling basic exceptions
H. Methods and Functional Programming
1. Parameter passing

2. Functional returns
3. Variable scope
4. Method overloading
5. Recursion
I. Object-Oriented Programming - Classes and Methods
1. Encapsulation of member data and methods
2. Instance vs. static members and methods
3. Data abstraction as realized through correct selection of member data and methods
4. Data privacy as supported by access modifiers, mutator, accessor and constructor methods
5. Object composition (the "has a" relationship)
6. Properties and indexers
7. Introduction to inner classes
8. Class organization using namespaces
J. Object-Oriented Programming – Inheritance
1. Introduction
2. Object hierarchy
3. Derived classes (the "is a" relationship)
4. Abstract classes and methods
5. Sealed classes and methods
6. Interfaces
7. Virtual methods
K. Intermediate GUI Elements
1. Understanding the Windows Forms control hierarchy
2. Adding controls to forms
3. Textbox controls
4. Command buttons
5. Checkboxes and listboxes
6. Other controls
7. Anchoring and docking
L. Arrays
1. Passing arrays to methods
2. Passing arrays by value and by reference
3. Multiple subscripted arrays
4. The foreach repetition structure
M. Error Reporting
1. Return values
2. Avoiding end-use output to report errors in functions or methods
3. The .NET exception hierarchy
4. Try and catch blocks
5. The finally block
6. Exception properties
7. Programmer designed exception classes
8. Handling overflows with the checked and unchecked operators
N. Algorithms
1. Role of algorithms in the problem-solving process
2. Simple uses of recursion for divide-and-conquer strategies
3. Simple sorting via a representative sort algorithm
4. Linear searches
O. Essential Examples and Assignment Areas
1. String/text processing
2. Numeric computation
3. User interaction through both console and GUI
4. Arrays
5. Using the .NET framework collections
6. Creating and using a programmer-defined class
P. Applications Used Throughout Course in Selected Areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Familiarization with the beginning-level lab environment
1. Modify and customize the settings of an Integrated Development Environment (IDE).
2. Use the IDE to create a new programming project.
3. Organize projects within an IDE to make submitting labs and switching project environments an orderly process.
4. Gain experience with the steps needed to edit a simple program.
5. Modify IDE settings to produce an industry standard code style.
B. Finding and fixing errors in simple programs
1. Demonstrate the complete edit-compile-run cycle of a simple program using IDE or command-line environment.
2. Distinguish between compiler/syntax errors and logic errors.
3. Develop strategies for dealing with each type of error.
4. Debug code to produce a working program.
C. Exploring the different data types using the compiler/IDE
1. Gain experience in effectively using the IDE to create code with C# value types.
2. Gain experience in effectively using the IDE to create code with C# reference types.
3. Use the IDE to assist in defining and using compound data types.
4. Solve syntax and logic problems that arise from typical incorrect formulation of data types.
D. Demonstrating user interaction (I/O) through the IDE's console or GUI capabilities
1. Play the role of user and programmer, alternately, to establish a user-interaction plan for a program.
2. Evaluate and comment on other students' user-interaction plan.
3. Change modes from source code design (editing mode) to end-user interaction (run mode) in the IDE in order to perform Q/A on the program.
4. Fix poor interaction behavior by adjusting source code and rerunning program until a satisfactory result is achieved.
E. Building a program that demonstrates "intelligence" though a combination of control statements
1. Become familiar with selection, loop and nesting to imbue a program with correct logic behavior.
2. Use structured programming to make control structures maintainable.
3. Run the program multiple times to verify that its control statements produce the correct behavior or output under any scenario.
4. Fix incorrect logic behavior by adjusting control structures and rerunning program until a satisfactory result is achieved.
F. Incorporating functions and class methods in programming projects
1. Gain experience in writing a function/method.
2. Use a previously written function or method in a client program.
3. Refine methods/functions by adding or changing their definitions and observe the result.
4. Deduce the impact of a function's or method's design on the programs that invoke it.
G. Building a program around object-oriented techniques
1. Use previously written classes to instantiate objects in program.
2. Use the IDE to assist in the creation of a programmer-defined class.
3. Demonstrate the correct choice of class members and methods for each class used.
4. Use the IDEs class view and object browser tools to navigate from one class to another within a program.
H. Exploring arrays
1. Understand the proper use of arrays.

2. Incorporate an array into a program to facilitate the solution of an assigned problem.
3. Investigate use of variable indices and loops to shorten and clarify the logic in programs.
4. Use debugging techniques to solve problems that arise during the testing of a program.
I. Devising and utilizing algorithms
1. Write a program that uses a combination of techniques, such as looping, arrays, logic and user I/O, all encapsulated in a coherent algorithm.
2. Test the algorithm by running the program multiple times giving it different initial values or inputs.
3. Implement a sorting or simple searching algorithm using arrays.
4. Transcribe an abstract algorithm into a concrete program that is written and tested using the IDE and submitted online for evaluation.

Special Facilities and/or Equipment

A. Access to a computer laboratory with C# compilers.
B. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor).
C. When taught via Foothill Global Access on the internet, a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Methods of Evaluation may include but are not limited to the following:

A. Tests and quizzes
B. Written laboratory assignments, which include source code, sample runs and documentation
C. Final examination

Method(s) of Instruction

Methods of Instruction may include but are not limited to the following:

A. Lectures, which include motivation for syntax and use of the C# language and OOP concepts, example programs, and analysis of these programs.
B. Online labs (for all sections, including those meeting face-to-face/on campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work.
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members.
C. Detailed review of programming assignments, which includes model solutions and specific comments on the student submissions.
D. In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs.
E. When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment.
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices.

Representative Text(s) and Other Materials

Doyle, Barbara. C# Programming: From Problem Analysis to Program Design. 5th ed. Course Technology Inc., 2016.
Heilsberg, et al. C# Programming Language. 4th ed. Addison Wesley, 2010.
Skeet, Jon. C# in Depth. 3rd ed. Manning, 2014.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science