STRANDS AND STANDARDS

COMPUTER PROGRAMMING, ADVANCED

Learning that works for Utah

CTE®

Course Description

This is an advanced course in computer programming/software engineering and applications. It reviews and builds on the concepts introduced in Computer Programming 1 and 2. It introduces students to dynamic data structures, advanced utilization of classes, and applications of recursion through the application of mathematical concepts. This course will also highlight the differences between the many different languages of computer programming.

Intended Grade Level	10-12
Units of Credit	1.0
Core Code	35.02.00.00.040
Concurrent Enrollment Core Code	35.02.00.13.040
Prerequisite	Computer Programming 2 or Teacher Approval
Skill Certification Test Number	840
Test Weight	1.0
License Area of Concentration	CTE and/or Secondary Education 6-12
Required Endorsement(s)	
Endorsement 1	Intro to Computer Science
Endorsement 2	Programming & Software Development
Endorsement 3	N/A

ADA COMPLIANT: March 2022

STRAND 1

Students will develop applications which make advanced use of the skills and concepts developed in Computer Programming 1 and Computer Programming 2.

Standard 1

Demonstrate the ability to develop complex applications.

- Develop complex applications using input, calculations, and output
- Develop complex applications using control structures (loops, if else, select, etc.)
- Develop complex applications in object-oriented programming
- Develop complex applications using data structures
- Develop complex applications using files (sequential files)

Standard 2

Utilize recursive algorithms.

- Analyze and solve recursive functions or methods
- Utilize recursive algorithms to solve a problem
- Identify the base case, recursive case, and action of each recursive function or method
- (Optional) Understand the use of a recursive helper function or method

Standard 3

Create advanced functions and methods.

- Create and use overloaded constructors and methods
- Create and use overloaded operators (C++)

Performance Skills

- Develop advanced applications using input, calculations, output, IF structures, iteration, sub-programs, recursion, arrays, sorting and a database.
- Demonstrate the ability to use random access files in a program.

STRAND 2

Students will use searching and sorting algorithms.

Standard 1

Demonstrate the ability to search data structures in programs.

- Develop a binary search
- Compare the efficiency and appropriateness of sequential and binary searches

Standard 2

Demonstrate the ability to sort data structures in programs.

- Sort arrays using iterative sorting algorithms (selection, insertion, bubble)
- Recognize recursive sorting algorithms (merge, quick, heap)
- Compare the efficiency of different sorting algorithms

Performance Skills

- Demonstrate the ability to search data structures using binary and hash searches comparing the efficiency between sequential and binary searches.
- Demonstrate the ability to sort data structures using quadratic (n2) and binary (n log n) sorts comparing the efficiency between various sorts using BigO notation.

STRAND 3

Students will utilize multidimensional arrays.

Standard 1

Utilize multidimensional arrays.

- Initialize multidimensional arrays
- Input and output data into and from multidimensional arrays
- Perform operations on multidimensional arrays
- Perform searches on multidimensional arrays

Performance Skills

STRAND 4

Students will properly employ dynamic data structures/ abstract data types (ADTs).

Standard 1

Demonstrate the ability to use stacks in programs.

- Declare stack structures
- Initialize stacks
- Check for empty and full stacks
- Push on to and pop off values from stacks
- Develop an application that utilizes stacks

Standard 2

Demonstrate the ability to use queues in programs.

- Declare queue structures
- Check for empty and full queues
- Initialize queues
- Enqueue values on to and dequeue values off of queues
- Develop an application that utilize queues

Performance Skills

Demonstrate the ability to use linked lists, stacks, queues, and binary trees.

STRAND 5

Students will design and implement advanced objected oriented concepts.

Standard 1

Implement object-oriented programs

- Create classes with minimal extraneous relationships (high cohesion and low coupling)
- Demonstrate and use composition and aggregation (HAS-A) relationships
- Demonstrate the use of class variables (static variables)

Standard 2

Implement inheritance in an objected oriented program.

- Utilize class hierarchies (parent-child relationships)
- Demonstrate IS-A relationships

- Override methods. Understand how to call the overriding method from the child
- Demonstrate the protected modifier
- Call a parent class constructor from the child's constructor

Standard 3
Create and use abstract classes.
- Create and implement abstract classes
- Implement interfaces (purely abstract classes)
- Know difference between abstract & interface classes

Standard 4
Implement polymorphism.
- Demonstrate that a parent object variable can hold an instance of a child class
- Determine IS-A relationships via code (e.g. instanceof, typeof, isa)
- Demonstrate typecasting via method calls of inherited objects

Performance Skills
- Develop advanced application projects.
- Develop advanced applications using object-oriented programming.
- Create user-defined inherited classes demonstrating overloading techniques.

STRAND 6
Students will use Unified Modeling Language (UML) to design object-oriented programs.

Standard 1
Demonstrate the use of an UML in design.
- Create an activity diagram
- Create a class diagram for the class hierarchy of a program
- Create a sequence diagram for a method
- Translate diagrams to code

STRAND 7
Students will develop a program of significant complexity as part of a portfolio.

Standard 1
Create an individual program of significant complexity.
- Create design documentation for the project
- Follow accepted object-oriented programming methodology when writing the code

Standard 2
Compile a portfolio of the individual and group programs developed.
- Include sample design work
- Include sample program source code

Performance Skills
- Create an individual program of significant complexity and size (300-500 lines).
- Compile a portfolio of the individual and group programs developed during the course.
- Participate in a work-based learning experience such as a job shadow, internship, field trip to a software

engineering firm or listen to an industry guest speaker and/or compete in a high school programming contest.

Workplace Skills

Workplace Skills taught:

- Communication
- Problem Solving
- Teamwork
- Critical Thinking
- Dependability
- Accountability
- Legal requirements / expectations

Skill Certificate Test Points by Strand

Test Name	Test #	Number of Test Points by Strand										Total Points	Total Questions
		1	2	3	4	5	6	7	8	9	10		
Computer Pro Advanced	840	2	5	3	7	12	5	0				34	30

Skills Reference Sheet

Assignment, Display, and Input	
`a = expression`	Evaluates `expression` and then assigns a copy of the result to the variable **a**.
`DISPLAY(expression)`	Displays the value of `(expression)` in the console window.
`INPUT( )`	Accepts a value from the user and returns the input value.
Arithmetic Operators and Numeric Procedures	

`a + b`
`a - b`
`a * b`
`a / b`
	The arithmetic operators `+`, `-`, `*`, and `/` are used to perform arithmetic on `a` and `b`.
	
	For example, `17 / 5` evaluates to `3.4`.
	
	The order of operations used in mathematics applies when evaluating expressions.

`a MODULUS b`
`-or- a`
`MOD b`
	Evaluates to the remainder when `a` is divided by `b`.
	
	For example, `17 MOD 5` evaluates to `2`.
	
	`MODULUS (MOD)` has the same precedence as the `*` and `/` operators.

Relational and Boolean Operators	
`NOT condition`	Evaluates to `true` if `condition` is `false`; otherwise evaluates to `false`.
`condition1 AND condition2`	Evaluates to `true` if both `condition1` and `condition2` are `true`; otherwise evaluates to `false`.
`condition1 OR condition2`	Evaluates to `true` if `condition1` is `true` or if `condition2` is `true` or if both `condition1` and `condition2` are `true`; otherwise evaluates to `false`.

`FOR(condition)`
`{`
`    <block of statements>`
`}`
	The code in `<block of statements>` is executed a certain number of times.

`WHILE(condition) { <block of statements> }`	The code in `<block of statements>` is repeated until the `(condition)` evaluates to `false`.
`IF(condition1) { <first block of statements> { ELSE IF(condition2) { <second block of statements> } ELSE { <third block of statements> }`	If `(condition1)` evaluates to `true`, the code in `<first block of statements>` is executed; if `(condition1)` evaluates to `false`, then `(condition2)` is tested; if `(condition2)` evaluates to `true`, the code in `<second block of statements>` is executed; if both `(condition1)` and `(condition2)` evaluate to `false`, then the code in `<third block of statements>` is executed.
Procedures and Procedure Calls	
`PROCEDURE procName( ) { <block of statements> }`	Defines `procName` as a procedure that takes no arguments. The procedure contains `<block of statements>`. The procedure `procName` can be called using the following notation: `procName( )`