Computer Programming, Advanced

Exam Information

Exam number
840

Items
30

Points
34

Prerequisites
Computer Programming 2
or Teacher Approval

Recommended course length
One Year

National Career Cluster
Information Technology

Performance standards
Included (Optional)

Certificate available
Yes

Vocabulary List
Yes, included

Description

The Computer Programming, Advanced industry certification exam assesses advanced concepts in computer programming and software engineering. It reviews and builds on the concepts introduced in Computer Programming 1 and 2, focusing on dynamic data structures, advanced utilization of classes, and applications of recursion using mathematical concepts. The exam also evaluates learners' understanding of the differences between various programming languages.

Exam Blueprint

Standard	Percentage of exam
1. Application Development	6%
2. Algorithms	15%
3. Multidimensional Arrays	9%
4. Dynamic Data Structures/Abstract Data Types	21%
5. Advanced Objected Oriented Concepts	35%
6. Unified Modeling Language (UML)	15%
7. Program Development	0%

Standard 1

Students will develop applications which make advanced use of the skills and concepts developed in computer programming 1 & 2.

Objective 1 Demonstrate the ability to develop complex applications.
1. Develop complex applications using input, calculations, and output.
2. Develop complex applications using control structures (loops, if else, select, etc.).
3. Develop complex applications in object-oriented programming.
4. Develop complex applications using data structures.
5. Develop complex applications using files (sequential files).

Objective 2 Utilize recursive algorithms.
1. Analyze and solve recursive functions or methods.
2. Utilize recursive algorithms to solve a problem.
3. Identify the base case, recursive case, and action of each recursive function or method.
4. (Optional) Understand the use of a recursive helper function or method.

Objective 3 Create advanced functions and methods.
1. Create and use overloaded constructors and methods.
2. Create and use overloaded operators (C++).

Standard 1 Performance Evaluation included below (Optional)

STANDARD 2

Students will use searching and sorting algorithms.

Objective 1 Demonstrate the ability to search data structures in programs.
1. Develop a binary search.
2. Compare the efficiency and appropriateness of sequential and binary searches.

Objective 2 Demonstrate the ability to sort data structures in programs.
1. Sort arrays using iterative sorting algorithms (selection, insertion, bubble).
2. Recognize recursive sorting algorithms (merge, quick, heap).
3. Compare the efficiency of different sorting algorithms.

STANDARD 3

Students will utilize multidimensional arrays.

Objective 1 Utilize multidimensional arrays.
1. Initialize multidimensional arrays.
2. Input and output data into and from multidimensional arrays.
3. Perform operations on multidimensional arrays.
4. Perform searches on multidimensional arrays.

Standard 3 Performance Evaluation included below (Optional)

STANDARD 4

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

Objective 1 Demonstrate the ability to use stacks in programs.
1. Declare stack structures.
2. Initialize stacks.
3. Check for empty and full stacks.
4. Push on to and pop off values from stacks.
5. Develop an application that utilizes stacks.

Objective 2 Demonstrate the ability to use queues in programs.
1. Declare queue structures.
2. Check for empty and full queues.
3. Initialize queues.
4. Enqueue values on to and dequeue values off of queues.
5. Develop an application that utilizes queues.

Standard 4 Performance Evaluation included below (Optional)

STANDARD 5

Students will design and implement advanced objected oriented concepts.

Objective 1 Implement object-oriented programs.
1. Create classes with minimal extraneous relationships (high cohesion and low coupling).

2. Demonstrate and use composition and aggregation (HAS-A) relationships.
3. Demonstrate the use of class variables (static variables).

Objective 2 Implement inheritance in an objected oriented program.

1. Utilize class hierarchies (parent-child relationships).
2. Demonstrate IS-A relationships.
3. Override methods. Understand how to call the overriding method from the child.
4. Demonstrate the protected modifier.
5. Call a parent class constructor from the child's constructor.

Objective 3 Create and use abstract classes.
1. Create and implement abstract classes.
2. Implement interfaces (purely abstract classes).
3. Know the difference between abstract & interface classes.

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

Standard 5 Performance Evaluation included below (Optional)

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

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

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

Objective 1 Create an individual program of significant complexity.
1. Create design documentation for the project.

2. Follow accepted object-oriented programming methodology when writing the code.

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

Standard 7 Performance Evaluation included below (Optional)

Computer Programming, Advanced

Performance assessments may be completed and evaluated at any time during the course. The following performance skills are to be used in connection with the associated standards and exam. To pass the performance standard the student must attain a performance standard average of 8 or higher on the rating scale. Students may be encouraged to repeat the objectives until they average 8 or higher.

Student's Name: _____

Class: _____

Performance standards rating scale

0	Limited skills	2	→	4	Moderate skills	6	→	8	High skills	10

Standard 1 – Application Development   Score:
- 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.

Standard 2 – Algorithms   Score:
- 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.

Standard 4 – Dynamic Data Structures/Abstract Data Types   Score:
- Demonstrate the ability to use linked lists, stacks, queues, and binary trees.

Standard 5 – Advanced Object-Oriented Concepts   Score:
- Develop advanced application projects.
- Develop advanced applications using object-oriented programming.

- Create user-defined inherited classes demonstrating overloading techniques.

Standard 7 – Program Development   Score:

- 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 listened to an industry guest speaker and/or competed in a high school programming contest.

Performance standard average score:

Evaluator Name: _____

Evaluator Title: _____

Evaluator Signature: _____

Date: _____

Computer Programming Advanced – 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.

	<u>Procedures and Procedure Calls</u>

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( )