Computer Science II

Subject: Career Development and Career and Technical Education
Grade: 11
Expectations: 59
Breakouts: 208

(a) Introduction.

1. Career and technical education instruction provides content aligned with challenging academic standards, industry-relevant technical knowledge, and college and career readiness skills for students to further their education and succeed in current and emerging professions
2. The Science, Technology, Engineering, and Mathematics (STEM) Career Cluster focuses on planning, managing, and providing scientific research and professional and technical services such as laboratory and testing services and research and development services
3. Computer Science II will foster students' creativity and innovation by presenting opportunities to design, implement, and present meaningful programs through a variety of media. Students will collaborate with one another, their instructor, and various electronic communities to solve the problems presented throughout the course. Through computational thinking and data analysis, students will identify task requirements, plan search strategies, and use computer science concepts to access, analyze, and evaluate information needed to solve problems. By using computer science knowledge and skills that support the work of individuals and groups in solving problems, students will select the technology appropriate for the task, synthesize knowledge, create solutions, and evaluate the results. Students will gain an understanding of computer science through the study of technology operations, systems, and concepts
4. Students are encouraged to participate in extended learning experiences such as career and technical student organizations and other leadership or extracurricular organizations
5. Statements that contain the word "including" reference content that must be mastered, while those containing the phrase "such as" are intended as possible illustrative examples

(b) Knowledge and Skills Statements

(1) Employability. The student identifies various employment opportunities in the computer science field. The student is expected to:

(A) identify job and internship opportunities and accompanying job duties and tasks and contact one or more companies or organizations to explore career opportunities;

(i) identify job opportunities

(ii) identify internship opportunities

(iii) identify accompanying job duties

(iv) identify accompanying job tasks

(v) identify [the] accompanying duties [of the identified internship opportunity]

(vi) identify [the] accompanying tasks [of the identified internship opportunity]

(vii) contact one or more companies or organizations to explore career opportunities

(B) examine the role of certifications, resumes, and portfolios in the computer science profession;

(i) examine the role of certifications in the computer science profession

- (ii) examine the role of resumes in the computer science profession
- (iii) examine the role of portfolios in the computer science profession

(C) employ effective technical reading and writing skills;

- (i) employ effective technical reading skills
- (ii) employ effective technical writing skills

(D) employ effective verbal and non-verbal communication skills;

- (i) employ effective verbal communication skills
- (ii) employ effective non-verbal communication skills

(E) solve problems and think critically;

- (i) solve problems
- (ii) think critically

(F) demonstrate leadership skills and function effectively as a team member;

- (i) demonstrate leadership skills
- (ii) function effectively as a team member

(G) identify legal and ethical responsibilities in relation to the field of computer science;

- (i) identify legal responsibilities in relation to the field of computer science
- (ii) identify ethical responsibilities in relation to the field of computer science

(H) demonstrate planning and time-management skills; and

- (i) demonstrate planning skills
- (ii) demonstrate time-management skills

(I) compare university computer science programs.

- (i) compare university computer science programs

(2) Creativity and innovation. The student develops products and generates new understandings by extending existing knowledge. The student is expected to:

(A) use program design problem-solving strategies to create program solutions;

- (i) use program design problem-solving strategies to create program solutions

(B) read, analyze, and modify programs and their accompanying documentation such as an application programming interface (API), internal code comments, external documentation, or readme files;

- (i) read programs
- (ii) read [a program's] accompanying documentation
- (iii) analyze programs
- (iv) analyze [a program's] accompanying documentation
- (v) modify programs
- (vi) modify [a program's] accompanying documentation

(C) follow a systematic problem-solving process that identifies the purpose and goals, the data types and objects needed, and the subtasks to be performed;

- (i) follow a systematic problem-solving process that identifies the purpose
- (ii) follow a systematic problem-solving process that identifies the goals
- (iii) follow a systematic problem-solving process that identifies the data types needed
- (iv) follow a systematic problem-solving process that identifies the objects needed
- (v) follow a systematic problem-solving process that identifies the subtasks to be performed

(D) compare design methodologies and implementation techniques such as top-down, bottom-up, and black box;

- (i) compare design methodologies techniques and implementation techniques

(E) trace a program, including inheritance and black box programming;

- (i) trace a program, including inheritance
- (ii) trace a program, including black box programming

(F) choose, identify, and use the appropriate abstract data type, advanced data structure, and supporting algorithms to properly represent the data in a program problem solution; and

- (i) choose the appropriate abstract data type to properly represent the data in a program problem solution
- (ii) choose the appropriate advanced data structure to properly represent the data in a program problem solution
- (iii) choose the appropriate supporting algorithms to properly represent the data in a program problem solution
- (iv) identify the appropriate abstract data type to properly represent the data in a program problem solution
- (v) identify the appropriate advanced data structure to properly represent the data in a program problem solution
- (vi) identify the appropriate supporting algorithms to properly represent the data in a program problem solution
- (vii) use the appropriate abstract data type to properly represent the data in a program problem solution
- (viii) use the appropriate advanced data structure to properly represent the data in a program problem solution
- (ix) use the appropriate supporting algorithms to properly represent the data in a program problem solution

(G) use object-oriented programming development methodology, including data abstraction, encapsulation with information hiding, inheritance, and procedural abstraction in program development.

- (i) use object-oriented programming development methodology, including data abstraction
- (ii) use object-oriented programming development methodology, including encapsulation with information hiding
- (iii) use object-oriented programming development methodology, including inheritance
- (iv) use object-oriented programming development methodology, including procedural abstraction in program development

(3) Communication and collaboration. The student communicates and collaborates with peers to contribute to his or her own learning and the learning of others. The student is expected to:

- (A) use the principles of software development to work in software design teams;
  - (i) use the principles of software development to work in software design teams
- (B) break a problem statement into specific solution requirements;
  - (i) break a problem statement into specific solution requirements
- (C) create a program development plan;
  - (i) create a program development plan
- (D) code part of a solution from a program development plan while a partner codes the remaining part;
  - (i) code part of a solution from a program development plan while a partner codes the remaining part
- (E) collaborate with a team to test a solution, including boundary and standard cases; and
  - (i) collaborate with a team to test a solution, including boundary cases
  - (ii) collaborate with a team to test a solution, including standard cases
- (F) develop presentations to report the solution findings.
  - (i) develop presentations to report the solution findings

(4) Data literacy and management. The student locates, analyzes, processes, and organizes data. The student is expected to:

- (A) use programming file structure and file access for required resources;
  - (i) use programming file structure for required resources
  - (ii) use programming file access for required resources
- (B) acquire and process information from text files, including files of known and unknown sizes;
  - (i) acquire information from text files, including files of known sizes
  - (ii) acquire information from text files, including files of unknown sizes
  - (iii) process information from text files, including files of known sizes
  - (iv) process information from text files, including files of unknown sizes
- (C) manipulate data using string processing;
  - (i) manipulate data using string processing
- (D) manipulate data values by casting between data types;
  - (i) manipulate data values by casting between data types
- (E) use the structured data type of one-dimensional arrays to traverse, search, modify, insert, and delete data;
  - (i) use the structured data type of one-dimensional arrays to traverse data
  - (ii) use the structured data type of one-dimensional arrays to search data
  - (iii) use the structured data type of one-dimensional arrays to modify data
  - (iv) use the structured data type of one-dimensional arrays to insert data

(v) use the structured data type of one-dimensional arrays to delete data

(F) identify and use the structured data type of two-dimensional arrays to traverse, search, modify, insert, and delete data;

(i) identify the structured data type of two-dimensional arrays to traverse data

(ii) identify the structured data type of two-dimensional arrays to search data

(iii) identify the structured data type of two-dimensional arrays to modify data

(iv) identify the structured data type of two-dimensional arrays to insert data

(v) identify the structured data type of two-dimensional arrays to delete data

(vi) use the structured data type of two-dimensional arrays to traverse data

(vii) use the structured data type of two-dimensional arrays to search data

(viii) use the structured data type of two-dimensional arrays to modify data

(ix) use the structured data type of two-dimensional arrays to insert data

(x) use the structured data type of two-dimensional arrays to delete data

(G) identify and use a list object data structure to traverse, search, insert, and delete data; and

(i) identify a list object data structure to traverse data

(ii) identify a list object data structure to search data

(iii) identify a list object data structure to insert data

(iv) identify a list object data structure to delete data

(v) use a list object data structure to traverse data

(vi) use a list object data structure to search data

(vii) use a list object data structure to insert data

(viii) use a list object data structure to delete data

(H) differentiate between categories of programming languages, including machine, assembly, high-level compiled, high-level interpreted, and scripted.

(i) differentiate between categories of programming languages, including machine

(ii) differentiate between categories of programming languages, including assembly

(iii) differentiate between categories of programming languages, including high-level compiled

(iv) differentiate between categories of programming languages, including high-level interpreted

(v) differentiate between categories of programming languages, including scripted

(5) Critical thinking, problem solving, and decision making. The student uses appropriate strategies to analyze problems and design algorithms. The student is expected to:

(A) develop sequential algorithms using branching control statements, including nested structures, to create solutions to decision-making problems;

(i) develop sequential algorithms using branching control statements, including nested structures, to create solutions to decision-making problems

(B) develop choice algorithms using selection control statements based on ordinal values;

- (i) develop choice algorithms using selection control statements based on ordinal values

(C) demonstrate the appropriate use of short-circuit evaluation in certain situations;

- (i) demonstrate the appropriate use of short-circuit evaluation in certain situations

(D) use Boolean algebra, including De Morgan's Law, to evaluate and simplify logical expressions;

- (i) use Boolean algebra, including De Morgan's Law, to evaluate logical expressions
- (ii) use Boolean algebra, including De Morgan's Law, to simplify logical expressions

(E) develop iterative algorithms using nested loops;

- (i) develop iterative algorithms using nested loops

(F) identify, trace, and appropriately use recursion in programming solutions, including algebraic computations;

- (i) identify recursion in programming solutions, including algebraic computations
- (ii) trace recursion in programming solutions, including algebraic computations
- (iii) appropriately use recursion in programming solutions, including algebraic computations

(G) trace, construct, evaluate, and compare search algorithms, including linear searching and binary searching;

- (i) trace search algorithms, including linear searching
- (ii) trace search algorithms, including binary searching
- (iii) construct search algorithms, including linear searching
- (iv) construct search algorithms, including binary searching
- (v) evaluate search algorithms, including linear searching
- (vi) evaluate search algorithms, including binary searching
- (vii) compare search algorithms, including linear searching and binary searching

(H) identify, describe, trace, evaluate, and compare standard sorting algorithms, including selection sort, bubble sort, insertion sort, and merge sort;

- (i) identify standard sorting algorithms, including selection sort
- (ii) identify standard sorting algorithms, including bubble sort
- (iii) identify standard sorting algorithms, including insertion sort
- (iv) identify standard sorting algorithms, including merge sort
- (v) describe standard sorting algorithms, including selection sort
- (vi) describe standard sorting algorithms, including bubble sort
- (vii) describe standard sorting algorithms, including insertion sort
- (viii) describe standard sorting algorithms, including merge sort
- (ix) trace standard sorting algorithms, including selection sort
- (x) trace standard sorting algorithms, including bubble sort

(xi) trace standard sorting algorithms, including insertion sort

(xii) trace standard sorting algorithms, including merge sort

(xiii) evaluate standard sorting algorithms, including selection sort

(xiv) evaluate standard sorting algorithms, including bubble sort

(xv) evaluate standard sorting algorithms, including insertion sort

(xvi) evaluate standard sorting algorithms, including merge sort

(xvii) compare standard sorting algorithms, including selection sort, bubble sort, insertion sort, and merge sort

(I) measure time and space efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for best, average, and worst-case data patterns;

(i) measure time efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for best-case data patterns

(ii) measure time efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for average-case data patterns

(iii) measure time efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for worst-case data patterns

(iv) measure space efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for best-case data patterns

(v) measure space efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for average-case data patterns

(vi) measure space efficiency of various sorting algorithms, including analyzing algorithms using "big-O" notation for worst-case data patterns

(J) develop algorithms to solve various problems such as factoring, summing a series, finding the roots of a quadratic equation, and generating Fibonacci numbers;

(i) develop algorithms to solve various problems

(K) test program solutions by investigating boundary conditions; testing classes, methods, and libraries in isolation; and performing stepwise refinement;

(i) test program solutions by investigating boundary conditions

(ii) test program solutions by testing classes in isolation

(iii) test program solutions by testing methods in isolation

(iv) test program solutions by testing libraries in isolation

(v) test program solutions by performing stepwise refinement

(L) identify and debug compile, syntax, runtime, and logic errors;

(i) identify compile errors

(ii) identify syntax errors

(iii) identify runtime errors

(iv) identify logic errors

(v) debug compile errors

(vi) debug syntax errors

(vii) debug runtime errors

(viii) debug logic errors

(M) compare efficiency of search and sort algorithms by using informal runtime comparisons, exact calculation of statement execution counts, and theoretical efficiency values using "big-O" notation, including worst-case, best-case, and average-case time/space analysis;

(i) compare efficiency of search and sort algorithms by using informal runtime comparisons, including worst-case analysis

(ii) compare efficiency of search and sort algorithms by using informal runtime comparisons, including best-case analysis

(iii) compare efficiency of search and sort algorithms by using informal runtime comparisons including average-case time/space analysis

(iv) compare efficiency of search and sort algorithms by using exact calculation of statement execution counts, including worst-case analysis

(v) compare efficiency of search and sort algorithms by using exact calculation of statement execution counts, including best-case analysis

(vi) compare efficiency of search and sort algorithms by using exact calculation of statement execution counts, including average-case time/space analysis

(vii) compare efficiency of search and sort algorithms by using theoretical efficiency values using "big-O" notation, including worst-case analysis

(viii) compare efficiency of search and sort algorithms by using theoretical efficiency values using "big-O" notation, including best-case analysis

(ix) compare efficiency of search and sort algorithms by using theoretical efficiency values using "big-O" notation, including average-case time/space analysis

(N) count, convert, and perform mathematical operations in the decimal, binary, octal, and hexadecimal number systems;

(i) count in the decimal number [system]

(ii) count in the binary number [system]

(iii) count in the octal number [system]

(iv) count in the hexadecimal number [system]

(v) convert in the decimal number [system]

(vi) convert in the binary number [system]

(vii) convert in the octal number [system]

(viii) convert in the hexadecimal number [system]

(ix) perform mathematical operations in the decimal number [system]

(x) perform mathematical operations in the binary number [system]

(xi) perform mathematical operations in the octal number [system]

(xii) perform mathematical operations in the hexadecimal number [system]

(O) identify maximum integer boundary, minimum integer boundary, imprecision of real number representations, and round-off errors;

(i) identify maximum integer boundary

(ii) identify minimum integer boundary

(iii) identify imprecision of real number representations

(iv) identify round-off errors

(P) create program solutions to problems using a mathematics library;

(i) create program solutions to problems using a mathematics library

(Q) use random number generator algorithms to create simulations;

(i) use random number generator algorithms to create simulations

(R) use composition and inheritance relationships to identify and create class definitions and relationships;

(i) use composition relationships to identify class definitions

(ii) use composition relationships to identify class relationships

(iii) use composition relationships to create class definitions

(iv) use composition relationships to create class relationships

(v) use inheritance relationships to identify class definitions

(vi) use inheritance relationships to identify class relationships

(vii) use inheritance relationships to create class definitions

(viii) use inheritance relationships to create class relationships

(S) explain and use object relationships between defined classes, abstract classes, and interfaces;

(i) explain object relationships between defined classes, abstract classes, and interfaces;

(ii) use object relationships between defined classes, abstract classes, and interfaces;

(T) create object-oriented class definitions and declarations using variables, constants, methods, parameters, and interface implementations;

(i) create object-oriented class definitions using variables

(ii) create object-oriented class definitions using constants

(iii) create object-oriented class definitions using methods

(iv) create object-oriented class definitions using parameters

(v) create object-oriented class definitions using interface implementations

(vi) create object-oriented class declarations using variables

(vii) create object-oriented class declarations using constants

(viii) create object-oriented class declarations using methods

(ix) create object-oriented class declarations using parameters

(x) create object-oriented class declarations using interface implementations

(U) create adaptive behaviors using polymorphism;

(i) create adaptive behaviors using polymorphism

(V) use reference variables for object and string data types;

(i) use reference variables for object data types

(ii) use reference variables for string data types

(W) use value and reference parameters appropriately in method definitions and method calls;

(i) use value parameters appropriately in method definitions

(ii) use value parameters appropriately in method calls

(iii) use reference parameters appropriately in method definitions

(iv) use reference parameters appropriately in method calls

(X) implement access scope modifiers;

(i) implement access scope modifiers

(Y) use object comparison for content quality;

(i) use object comparison for content quality

(Z) duplicate objects using the appropriate deep or shallow copy;

(i) duplicate objects using the appropriate deep or shallow copy

(AA) apply functional decomposition to a program solution;

(i) apply functional decomposition to a program solution

(BB) create objects from class definitions through instantiation; and

(i) create objects from class definitions through instantiation

(CC) examine and mutate the properties of an object using accessors and modifiers.

(i) examine the properties of an object using accessors and modifiers

(ii) mutate the properties of an object using accessors and modifiers