CIEP Submission Form

Computer Science (6-12)

(for Educator Preparation Chapter adopted 8-12-2021)

Institution Name:
Date Submitted:

Program Level: *Select one of the options below.*

☐Class B
☐Alternative Class A

Submitting for: *Choose one of the options below.*

☐Initial review of a proposed program
☐Continuing review of a currently approved program
☐Resubmission to address unmet standards and/or conditions

Overview of Each Required Section:

I. **Background Information:** Provide background information about the program (checklist; numbers of admissions, completers, and recommendations for certification). The "n"s reported here are used to determine if "n"s reported in data tables are consistent.

II. **Key Assessments, Data, and Data Analysis:** Provide an overview of the key assessment in the Section II chart. Key Assessments are typically summative assessments of candidate proficiencies. For each key assessment, included the completed coversheet; assessment instrument, instructions, or test specification information; rubric or scoring guide; and data table(s). Program faculty preparing submissions should use the Rubric for Key Assessments.

III. **Alignment of Standards to Curriculum and Key Assessments:** Provide an overview of how the program ensures each indicator is adequately addressed in curriculum and key assessments so reviewers know where to look to for evidence. Reviewers use the course descriptions and assessment documents, not the chart, to determine whether each indicator is adequately addressed.

IV. **Summary of Field Experiences Prior to Internship:** Provide an overview of how the program requires candidates to demonstrate developing proficiencies in field experiences prior to internship. Copies of instructions or assignments must be submitted. Assessment information is not required but may be submitted. Field experiences should have clear purposes and reflect increasing expectations. Program faculty preparing submissions should use the Rubric for Field Experiences Prior to Internship.

SECTION I Background Information

1. Include the proposed checklist as a separate document.

2. Data on Unconditional Admissions, Program Completers, and Certificates Issued
Programs should report at least three years of data. If the "n" over three years is less than 10, the program should report five years of data.

Academic Year September 1 to August 31	**Number of Unconditional Admissions**	**Number of Program Completers**[1]	**Number Recommended for Alabama Certification**

[1] Use the Title II definition for program completers.

SECTION II   Key Assessments, Data, and Data Analysis

1. *Assessments #1-#5 are required. No more than eight key assessments may be submitted.*
2. *Complete a coversheet for each key assessment and attach it to the instrument or instructions, or test specifications; rubric or scoring guide; and data tables(s).   Submit these documents in a Key Assessments folder on the flash drive and a section of the binder.*

#	Key Assessment Title	Name of Key Assessment[2]	Type of Key Assessment[3]	When Required by Program[4]
	State Certification Tests:[5]			
1 a	**Praxis Computer Science**		State Certification Tests	
1 b	**Praxis PLT**			
2	**Content Knowledge**[6]			
3	**Planning Instruction**[7]			
4	**Internship**			
5	**Effect on Student Learning**[8]			
6[9]				
7				
8				

[2] Identify assessment by title used in the program.
[3] Types of assessment include but are not limited to essay, case study, project, comprehensive exam, reflection, state certification test, and portfolio.
[4] Assessments might be required at the time of admission to the program, admission to internship, during a required course, or at program completion.
[5] Test data must include the percentage of candidates who passed the tests for the last three years. Total scores and appropriate sub-test data must be reported.
[6] Examples of appropriate content knowledge assessments include grade analyses, comprehensive examinations, portfolio tasks, and culminating performances.
[7] Examples of appropriate assessments for planning instruction include developing lesson or unit plans that address the breadth and depth of the teaching field, individualized education plans, needs assessments, or intervention plans.
[8] Examples of appropriate assessments for effect on student learning include those based on samples of student work, portfolio tasks, case studies, and appropriate follow-up studies.
[9] Examples of optional assessments addressing program standards include but are not limited to evaluations of field experiences, case studies, specific portfolio artifacts, complete portfolios, and follow-up studies.

SECTION III Alignment of Standards to Curriculum and Key Assessments

Identify the curriculum components and key assessments listed in Section II that address the standard and indicators. Only courses that directly address indicators should be listed. In most cases, an indicator will be addressed by more than one key assessment. Cross-references to the standards and indicators should be inserted into the assessment instruments, scoring guides, and data tables.

Standard 1 Knowledge.
Prior to program completion, prospective teachers of computer science shall demonstrate knowledge sufficient to teach content related to:

Indicators	**Curriculum Components—Courses or Other Requirements**[10] *(Include course prefix, number, and name.)*	**Key Assessment(s)** *(Identify by key assessment number[s] in Section II.)*
1.1 Impacts of Computing.		
1.1.1 Impact of, obstacles to, and effects of computing.		
1.1.2 Issues regarding intellectual property, ethics (e.g., concerns related to artificial intelligence and machine learning capabilities that may affect society), privacy, and security in computing.		
1.2 Algorithms and Computational Thinking.		
1.2.1 Abstraction; pattern recognition in data samples and computational processes; problem decomposition; and number base conversion.		
1.2.2 Algorithm analysis, searching and sorting algorithms, recursive algorithms, randomization, and algorithm expression (e.g., pseudocode and flowcharts).		
1.3 Programming.		
1.3.1 Programming control structures, standard operators (e.g., arithmetic, relational and logical), variables, correctness, extensibility, modifiability, and reusability.		

1.3.2 Procedures, function, and methods; event-driven programs; usability; data structures (e.g., stacks, queues, lists); debugging; documenting and reviewing code; libraries and application programming interfaces (APIs), integrated development environments (IDEs); and programming language paradigms, including object-oriented concepts.		
1.4 Data.		
1.4.1 Digitalization of information; data encryption and decryption; data compression, error detection and correction; and computational tools.		
1.4.2 Simulation, modeling, and manipulation of data.		
1.5 Computing Systems and Networks.		
1.5.1 Hardware and software for designing systems, identifying and fixing problems, and troubleshooting issues), software life cycle, operating systems, computing systems, virtual machines, communication between devices, and cloud computing.		
1.5.2 Networks, including protocols, encryption, and security issues and the Web.		

Standard 2 Abilities. Prior to program completion, prospective teachers of computer science shall demonstrate ability to teach students to:		
Indicators	**Curriculum Components—Courses or Other Requirements** *(Include course prefix, number, and name.)*	**Key Assessment(s)** *(Identify by key assessment number[s] in Section II.)*
2.1 Computing Systems.		
2.1.1 Recommend improvements to the design of computing devices, based on an analysis of how users interact with software and hardware devices.		
2.1.2 Design projects that combine hardware and software components to collect, process, and output data.		
2.1.3 Systematically identify and fix problems with computing devices and their components.		
2.1.4 Explain how abstractions hide the underlying implementation details of computing systems embedded in everyday objects.		
2.1.5 Compare levels of abstraction and interactions between application software, system software, and hardware layers.		
2.1.6 Develop guidelines that convey systematic trouble-shooting strategies that others can use to identify and fix errors.		
2.1.7 Categorize the roles of operating system software.		
2.1.8 Illustrate ways computing systems implement logic, input, and output through hardware components.		

2. Networks and the Internet.		
2.2.1 Model the role of protocols in transmitting data across networks and the Internet. (6-8)		
2.2.2 Explain how physical and digital security measures protect electronic information, including public key cryptography.		
2.2.3 Apply multiple methods of encryption to model the secure transmission of information.		
2.2.4 Evaluate the scalability and reliability of networks, by describing the relationship between routers, switches, servers, topology, and addressing.		
2.2.5 Give examples to illustrate how sensitive data can be affected by malware and other attacks.		
2.2.6 Recommend security measures to address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.		
2.2.7 Compare various security measures, considering tradeoffs between the usability and security of a computing system.		
2.2.8 Explain tradeoffs when selecting and implementing cybersecurity recommendations.		
2.2.9 Describe the issues that impact network functionality (e.g., bandwidth, load, delay, topology).		
2.2.10 Compare ways software developers protect devices and information from unauthorized access, considering different classifications of intrusion prevention systems and how each identifies malicious activity, logs information about the activity, reports it, and attempts to block or stop it.		

3. Data and Analysis.		
2.3.1 Represent data using multiple encoding schemes.		
2.3.2 Collect data using computational tools and transform the data to make the data more useful and reliable.		
2.3.3 Refine computational models based on the data they have generated.		
2.3.4 Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.		
2.3.5 Evaluate the tradeoffs in how data elements are organized and where data are stored.		
2.3.6 Create interactive data visualizations using software tools to help others better understand real-world phenomena.		
2.3.7 Create computational models that represent the relationships among different elements of data collected from a phenomena or process.		
2.3.8 Use data analysis tools and techniques to identify patterns in data representing complex systems.		
2.3.9 Select data collection tools and techniques to generate data sets that support a claim or communicate information.		
2.3.10 Evaluate the ability of models and simulations to test and support the refinement of hypotheses.		
4. Algorithms and Programming.		
2.4.1 Use flowcharts and/or pseudocode to address complex problems as algorithms.		
2.4.2 Create clearly named variables that represent different data types and perform operations (e.g.,		

arithmetic, relational, and logical operations) on their values.		
2.4.3 Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.		
2.4.4 Decompose problems and subproblems into parts (functions) to facilitate the design, implementation, and review of programs.		
2.4.5 Create procedures with parameters to organize code ad make it easier to reuse.		
2.4.6 Seek and incorporate feedback from team members and users to refine a solution that meets user needs.		
2.4.7 Incorporate existing code, media, and libraries into original programs, and give attribution.		
2.4.8 Systematically test and refine programs using a range of test cases.		
2.4.9 Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.		
2.4.10 Document programs in order to make them easier to follow, test, and debug.		
2.4.11 Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.		
2.4.12 Use lists or arrays to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.		
2.4.13 Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.		

2.4.14 Design and iteratively develop computational artifacts for practical intent, personal expression, or to address a societal issue by using events to initiate instructions.		
2.4.15 Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and objects.		
2.4.16 Create artifacts by using procedures within a program, combinations of data and procedures, or independent but interrelated programs.		
2.4.17 Systematically design and develop programs for broad audiences by incorporating feedback from users.		
2.4.18 Evaluate licenses that limit or restrict use of computational artifacts when using resources such as libraries.		
2.4.19 Evaluate and refine computational artifacts to make them more usable and accessible.		
2.4.20 Design and develop computational artifacts working in team roles using collaborative tools and pair programming techniques.		
2.4.21 Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.		
2.4.22 Demonstrate ways a given algorithm applies to problems across disciplines.		
2.4.23 Describe how artificial intelligence drives many software and physical systems.		
2.4.24 Implement an artificial intelligence algorithm to play a game against a human opponent or solve a problem.		

2.4.25 Use and adapt classic algorithms (e.g., shortest path, sorting, and searching) to solve computational problems.		
2.4.26 Evaluate algorithms in terms of their efficiency, correctness, and clarity.		
2.4.27 Compare and contrast fundamental data structures and their uses.		
2.4.28 Illustrate the flow of execution of a recursive algorithm.		
2.4.29 Construct solutions to problems using student-created components, such as procedures, nodules and objects.		
2.4.30 Analyze a large-scale computational problem and identify generalizable patterns that can be applied to a solution.		
2.4.31 Demonstrate code reuse by creating programming solutions using libraries and APIs.		
2.4.32 Plan and develop programs for broad audiences using a software life-cycle process.		
2.4.33 Explain security issues that might lead to compromised computer programs.		
2.4.34 Develop programs for multiple computing platforms.		
2.4.35 Use version control systems, integrated development environments (IDEs), and collaborative tools and practices (code documentation) in a group software project.		
2.4.36 Develop and use a series of test cases to verify that a program performs according to its design specifications.		

2.4.37 Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).		
2.4.38 Evaluate key qualities of a program through a process such as a code review.		
2.4.39 Modify an existing program to add additional functionality and discuss intended and unintended implications (e.g., breaking other functionality).		
2.4.40 Evaluate key qualities of a program through a process such as a code review.		
2.4.41 Compare multiple programming languages and discuss how their features make them suitable for solving different types of problems.		
5. Impacts of Computing.		
2.5.1 Compare tradeoffs associated with computing technologies that affect people's everyday activities and career options.		
2.5.2 Discuss issues of bias; accessibility for all users, including those with special needs; and usability in the design of existing technologies.		
2.5.3 Collaborate with contributors through strategies such as crowdsourcing or surveys when creating a computational artifact.		
2.5.4 Describe tradeoffs between allowing information to be public and keeping information private and secure, recognizing that nothing posted online is private.		
2.5.5 Evaluate the ways computing impacts personal, ethical, social, economic, and cultural practices.		
2.5.6 Test and refine computational artifacts to reduce bias and equity deficits.		

2.5.7 Demonstrate ways a given algorithm applies to problems across disciplines.		
2.5.8 Use tools and methods for collaboration on a project to increase connectivity of people in different cultures and career fields.		
2.5.9 Explain the beneficial and harmful effects that intellectual property laws can have on innovation.		
2.5.10 Explain the privacy concerns related to the collection and generation of data through automated processes that may not be evident to users.		
2.5.11 Evaluate the social and economic implications of privacy in the context of safety, law, and ethics.		
2.5.12 Evaluate computational artifacts to maximize their beneficial effects and minimize harmful effects on society.		
2.5.13 Evaluate the impact of equity, access, and influence on the distribution of computing resources in the global society.		
2.5.14 Predict how computational innovations that have revolutionized aspects of our culture might evolve.		
2.5.15 Debate laws and regulations that impact the development and use of software.		
2.5.16 Consider the impact of professional societies (e.g., Association for Computing Machinery, Institute of Electrical and Electronics Engineers, Association of Information.		

Standard 3 Pedagogy. Prior to program completion, prospective computer science teachers demonstrate ability to:		
Indicators	**Curriculum Components—Courses or Other Requirements** *(Include course prefix, number, and name.)*	**Key Assessment(s)** *(Identify by key assessment number[s] in Section II.)*
3.1 Encourage students from underrepresented groups to take computer science courses.		
3.2 Make students aware of trends in the computer science job market (e.g., emerging skills sets, entry requirements, career paths, and salaries).		
3.3 Use a variety of instructional strategies, including digital and physical (offline or unplugged) environments.		
3.4 Adapt instruction to student interests and abilities.		
3.5 Incorporate collaboration into instruction.		

Standard 4 Professionalism. Prior to program completion, prospective computer science teachers demonstrate ability to:		
Indicators	**Curriculum Components—Courses or Other Requirements** *(Include course prefix, number, and name.)*	**Key Assessment(s)** *(Identify by key assessment number[s] in Section II.)*
4.1 Articulate why all students are capable of learning computer science.		
4.2 Develop computer science curricula.		
4.3 Stay current with research on computer science education, including pedagogy and assessment.		
4.4 Learn collaboratively with other computer science teachers.		

SECTION IV   Summary of Field Experiences Prior to Internship

1. **List all courses (or other curriculum requirements) that have a required field experience, <u>in the order</u> that the courses are typically taken.** ***Include the course prefix, number, and title.***

Course Prefix	Course Number	Course Title

2. **Are field experiences always done in this order?**   **☐Yes**   **☐No**
   If no, provide a brief explanation.

3. **Briefly explain how placements are made to ensure that candidates are placed in diverse schools.**

4. **For each field experience, complete a field experience coversheet and attach it to the instructions or assignments for the field experience. Submit these in a Field Experience folder on the flash drive and a section in the binder.**