Arizona State University (ASU) CSE240 Introduction to Programming Languages Midterm Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Questions

1. How is an array defined in programming?

   A. A collection of different data types

   B. A single value stored in memory

   C. A collection of elements of the same data type stored contiguously

   D. A set of functions organized in a class

2. How is polymorphism defined in programming?

   A. The ability to instantiate multiple classes from a single base class

   B. A method that enables data sharing between classes

   C. The ability of different classes to be treated as instances of the same class through a common interface

   D. The act of combining two or more classes into one

3. What does the term "regression testing" refer to?

   A. Testing new functionality

   B. Testing after bug fixes to ensure old features still work

   C. Testing for performance under load

   D. Testing user interfaces

4. What is a data structure?

   A. A method of executing algorithms in programming

   B. A way to organize and store data for efficient access and modification

   C. A type of programming language used for data manipulation

   D. A collection of operations that can be performed on data

5. Which modifiers can be used to modify primitive data types in C++?

   A. static, volatile, const

   B. signed, unsigned, short, long

   C. integer, decimal, character

   D. public, private, protected

6. What advantage does inheritance provide in programming?

A. It allows for greater efficiency by reusing common code

B. It enables multiple classes to share the same data structure

C. It makes encapsulated code easier to read

D. It provides a way to alter the original class

7. What is inheritance in the context of programming?

A. A feature that allows a derived class to inherit properties from a base class

B. A method that changes the behavior of the base class

C. A structure that holds multiple data types simultaneously

D. A process of hiding the implementation details from the user

8. What is the purpose of defining a function in programming?

A. To create user interfaces

B. To execute a sequence of loop iterations

C. To perform a specific task when called

D. To store multiple variables

9. Which paradigm emphasizes simpler semantics and computation expressions in terms of mathematical functions?

A. Imperative

B. Functional

C. Logic

D. Object Oriented

10. Does logic programming divide the program into functions or procedures?

A. True

B. False

C. It depends on the implementation

D. Only in certain languages

Answers

1. C
2. C
3. B
4. B
5. B
6. A
7. A
8. C
9. B
10. B

Explanations

1. How is an array defined in programming?

A. A collection of different data types

B. A single value stored in memory

C. A collection of elements of the same data type stored contiguously

D. A set of functions organized in a class

An array is defined as a collection of elements of the same data type stored contiguously in memory. This means that all the elements within an array occupy adjacent memory locations, allowing for efficient access and manipulation of the data. The uniformity of data types within an array is crucial because it enables the programming language's memory management system to handle the data more effectively, performing operations such as indexing and looping without the need for additional type checks. Elements can be accessed using their index, making the retrieval of values direct and fast. Contiguity in memory is beneficial for performance reasons, as it enhances cache utilization—when one element is accessed, the nearby elements are likely to be loaded into the cache as well, leading to improved execution speed. This is a foundational concept in programming and is widely applied in various areas, such as algorithms and data structure design. In contrast, defining an array as a collection of different data types would contradict the fundamental nature of arrays in most programming languages, which require uniformity among the data types. Additionally, a single value stored in memory does not describe the concept of an array, as an array encompasses multiple elements. Lastly, a set of functions organized in a class pertains to the concepts of object-oriented programming and

2. How is polymorphism defined in programming?

A. The ability to instantiate multiple classes from a single base class

B. A method that enables data sharing between classes

C. The ability of different classes to be treated as instances of the same class through a common interface

D. The act of combining two or more classes into one

Polymorphism in programming is defined as the ability of different classes to be treated as instances of the same class through a common interface. This concept allows methods to use objects of different classes interchangeably, as long as these classes share a common interface or base class. By leveraging polymorphism, programs can be designed to handle objects of different types in a uniform way, thus enhancing flexibility and reusability of code. For instance, if multiple classes implement a specific method defined in an interface, any code that utilizes that interface can invoke the method on any of the implementing classes without knowing the specific type of object it is dealing with. This is especially useful in scenarios like collections or frameworks where the exact type of the objects is not known until runtime. The other options touch on related concepts, but they don't capture the essence of polymorphism as accurately. The ability to instantiate multiple classes from a single base class pertains more to inheritance, while data sharing between classes focuses on collaboration, and combining classes is related to class composition or inheritance but does not convey the concept of treating different objects uniformly.

3. What does the term "regression testing" refer to?

A. Testing new functionality

B. Testing after bug fixes to ensure old features still work

C. Testing for performance under load

D. Testing user interfaces

The concept of regression testing is specifically focused on validating that existing functionality remains intact after changes have been made to the code, such as bug fixes or new feature implementations. Whenever a modification takes place, whether it's correcting a defect or enhancing the software, regression testing is executed to confirm that the previously functioning aspects of the application are still operating as intended. This form of testing acts as a safeguard against introducing new errors into the already tested codebase and detects unintended side effects that might have arisen due to the recent changes. While other testing types relate to specific objectives, such as validating new features, performance under load, or usability of interfaces, they do not capture the essence of regression testing, which is uniquely concerned with maintaining the integrity of existing functions throughout the software development lifecycle.

4. What is a data structure?

A. A method of executing algorithms in programming

B. A way to organize and store data for efficient access and modification

C. A type of programming language used for data manipulation

D. A collection of operations that can be performed on data

A data structure is fundamentally defined as a way to organize and store data in a computer so that it can be accessed and modified efficiently. This means that the selection and arrangement of elements within a data structure directly influence how quickly and easily operations such as searching, inserting, deleting, and updating can be performed on the data. For instance, consider how arrays, linked lists, stacks, and queues each provide different ways of organizing data. Each structure offers its own strengths and weaknesses depending on the operations being performed and the nature of the data being stored. By choosing the appropriate data structure, programmers can optimize both the performance and efficiency of their programs, which is essential for handling large amounts of data or performing complex operations. Understanding this concept helps in making informed decisions about how to handle data in software development, which is crucial for achieving the desired performance and maintainability of a system.

5. Which modifiers can be used to modify primitive data types in C++?

A. static, volatile, const

B. signed, unsigned, short, long

C. integer, decimal, character

D. public, private, protected

The correct choice includes modifiers that specifically change the representation or characteristics of primitive data types in C++. In C++, primitive data types such as integers and floating-point numbers can be modified using signed, unsigned, short, and long. When using these modifiers, they allow for a more specific definition of the type. For instance, "signed" indicates that the variable can hold both positive and negative values, whereas "unsigned" allows the variable to hold only non-negative values, effectively doubling the positive range of values that can be stored. The "short" modifier specifies a smaller size of the integer type, which can help save memory, while "long" increases the size to accommodate larger values. These modifiers are particularly important for managing the limitations and capabilities of primitive types in applications. In contrast, the other choices include modifiers that either do not apply to primitive types (such as access specifiers like public, private, and protected) or refer to types that are not valid in C++ (like integer, decimal, and character as modifiers). Thus, choice B is accurate as it specifically cites modifiers that are utilized for modifying primitive data types in C++.

6. What advantage does inheritance provide in programming?

A. It allows for greater efficiency by reusing common code

B. It enables multiple classes to share the same data structure

C. It makes encapsulated code easier to read

D. It provides a way to alter the original class

Inheritance is a fundamental concept in object-oriented programming that primarily allows for code reuse. This practice enhances efficiency by enabling a derived class to inherit attributes and methods from a base class, which minimizes redundancy. By reusing existing code, developers can create new functionality without the need to rewrite common pieces of logic, thus leading to a more organized and maintainable codebase. This reuse of code can significantly speed up the development process and reduce the likelihood of introducing bugs, as the inherited code can be thoroughly tested and modified independently of the derived classes. Additionally, inheritance structures can promote more logical organization of classes in a hierarchy, making it easier for developers to understand the relationships between different entities within their code. It encourages the DRY (Don't Repeat Yourself) principle, which is crucial for effective programming practices. While other options might present interesting aspects of programming, they do not directly capture the primary advantage that inheritance offers as effectively as the reusability of common code does.

7. What is inheritance in the context of programming?

A. A feature that allows a derived class to inherit properties from a base class

B. A method that changes the behavior of the base class

C. A structure that holds multiple data types simultaneously

D. A process of hiding the implementation details from the user

Inheritance in programming is a fundamental concept in object-oriented programming (OOP) that allows a class, known as a derived class or subclass, to acquire properties and behaviors (methods) from another class, referred to as the base class or superclass. This mechanism enables code reusability and establishes a relationship between the classes, forming a hierarchy. When a derived class inherits from a base class, it can use the attributes and methods of the base class as if they were its own. This means that the derived class can extend or customize the inherited functionality as needed while maintaining a logical connection to the base class. For example, if a base class `Animal` has a method `makeSound`, a derived class `Dog` can inherit this method and may also override it to provide a specific implementation, such as barking. The other options describe different concepts in programming. One talks about altering the behavior of a class, which relates to polymorphism rather than inheritance. Another option refers to data structures that hold multiple types, like tuples or arrays, which does not involve inheritance. The final option touches on encapsulation, which is about restricting access to certain details in order to protect the integrity of an object, not about inheriting properties from one class to another

8. What is the purpose of defining a function in programming?

A. To create user interfaces

B. To execute a sequence of loop iterations

C. To perform a specific task when called

D. To store multiple variables

Defining a function in programming serves the fundamental purpose of encapsulating a specific task or operation that can be executed whenever it is called. Functions allow programmers to write reusable and modular code, which enhances readability and maintainability. When a function is defined, it can take parameters, perform computations or actions, and return a result. This modular approach means that you can write a piece of code once and use it in multiple places within a program without the need to rewrite the same logic each time. By using functions, developers can break down complex problems into simpler, manageable pieces, improving the overall structure of the code. This also facilitates easier debugging and testing, as functions can be tested in isolation. The other options, while relevant to programming, do not accurately capture the core purpose of a function. Functions do not inherently relate to user interfaces, executing loop iterations, or storing multiple variables, as these tasks can be accomplished through other constructs and paradigms in programming. Functions are specifically designed to perform well-defined tasks when invoked, making them a key construct in programming languages.

9. Which paradigm emphasizes simpler semantics and computation expressions in terms of mathematical functions?

A. Imperative

B. Functional

C. Logic

D. Object Oriented

The correct choice highlights the nature of functional programming, which is characterized by its focus on mathematical functions and the use of expressions to define computations. In functional programming, the fundamental building blocks are functions that take input values and produce output in a clear and predictable manner. This paradigm emphasizes the evaluation of expressions and the creation of more straightforward semantics through immutability and first-class functions. Functional programming allows for a high level of abstraction, where functions can be composed and reused, enabling developers to express ideas succinctly without concern for the underlying state or changes over time, which can lead to simpler and more maintainable code. By concentrating on the "what" rather than the "how" of computing processes, functional programming helps to avoid side effects and promotes a clearer understanding of the program's flow. In contrast, the other paradigms each have distinct features that do not prioritize the same level of simplicity and mathematical function emphasis. For instance, imperative programming focuses on detailed instructions and state changes, logic programming relies on defining relationships and rules, while object-oriented programming emphasizes encapsulation and interaction between objects.

10. Does logic programming divide the program into functions or procedures?

A. True

B. False

C. It depends on the implementation

D. Only in certain languages

Logic programming fundamentally differs from imperative programming paradigms, which typically structure programs around functions or procedures. In logic programming, the focus is on expressing facts and rules about problems rather than on describing a sequence of operations to be performed. This means that logic programming does not rely on the concept of functions or procedures; instead, it operates through logical statements that define relationships and implications. In languages like Prolog, for example, you define relationships and rules, and the logic engine processes these to answer queries, rather than executing a series of commands as in functional or procedural programming. As a result, the structure of logic programming does not divide the program into functions or procedures, supporting the assertion that the statement is false.