* \_\_\_\_\_\_\_ is a step-by-step method for solving a problem or doing a task.
* A construct
* A recursion
* An iteration
* **An algorithm**
* There are \_\_\_\_\_\_ basic constructs in computer -science.
* one
* two
* **three**
* four
* The \_\_\_\_\_\_\_construct tests a condition.
* sequence
* **decision**
* repetition
* flow
* The \_\_\_\_\_\_\_ construct uses a set of actions one after another.
* **sequence**
* decision
* repetition
* flow
* The \_\_\_\_\_\_\_ construct handles repeated actions.
* sequence
* decision
* **repetition**
* flow
* \_\_\_\_\_\_\_ is a pictorial representation of an algorithm.
* **A UML diagram**
* A program
* Pseudocode
* An algorithm
* \_\_\_\_\_\_\_ is an English-language-like representation of code.
* A UML diagram
* A program
* **Pseudocode**
* An algorithm
* \_\_\_\_\_\_\_ is a basic algorithm that adds a list of numbers.
* **Summation**
* Product
* Smallest
* Largest
* \_\_\_\_\_\_\_ is a basic algorithm that multiplies a list of numbers.
* Summation
* **Product**
* Smallest
* Largest
* \_\_\_\_\_\_\_ is a basic algorithm that arranges data according to its value.
* Inquiry
* **Sorting**
* Searching
* Recursion
* The items are divided into two lists (sorted and unsorted) \_\_\_\_\_ sort.
* only in a selection
* only in a bubble
* only in an insertion
* **in selection, bubble, or insertion**
* In \_\_\_\_\_\_\_ sort, the item that goes into the sorted list is always the first item in the unsorted list.
* selection
* bubble
* **insertion**
* every
* In \_\_\_\_\_\_\_ sort, the smallest item from the unsorted list is swapped with the item at the beginning of the unsorted list.
* **selection**
* bubble
* insertion
* every
* In \_\_\_\_\_\_ sort, the smallest item moves to the beginning of the unsorted list. There is no one-to-one swapping.
* selection
* bubble
* **insertion**
* every
* \_\_\_\_\_\_\_ is a basic algorithm in which we want to find the location of a target in a list of items.
* Sorting
* **Searching**
* Product
* Summation
* We use a \_\_\_\_\_\_\_ search for an unordered list.
* **sequential**
* binary
* bubble
* insertion
* We use a \_\_\_\_\_\_\_ search for an ordered list.
* sequential
* **binary**
* bubble
* insertion
* \_\_\_\_\_\_\_ is a process in which an algorithm calls itself.
* Insertion
* Searching
* **Recursion**
* Iteration
* The only language understood by computer hardware is a \_\_\_\_\_\_\_ language.
* **machine**
* symbolic
* high-level
* natural
* C, C++, and Java can be classified as \_\_\_\_\_\_\_ languages.
* machine
* symbolic
* **high-level**
* natural
* FORTRAN is a(n) \_\_\_\_\_\_\_ language.
* **procedural**
* functional
* declarative
* object-oriented
* Pascal is a(n) \_\_\_\_\_\_\_ language.
* **procedural**
* functional
* declarative
* object-oriented
* Java is a(n) \_\_\_\_\_\_\_ language.
* procedural
* functional
* declarative
* **object-oriented**
* LISP is a(n) \_\_\_\_\_\_\_ language.
* procedural
* functional
* **declarative**
* object-oriented
* \_\_\_\_\_\_\_ is a common language in the business -environment.
* FORTRAN
* C++
* C
* **COBOL**
* \_\_\_\_\_\_\_ is a popular object-oriented language.
* FORTRAN
* COBOL
* **Java**
* LISP
* A \_\_\_\_\_\_\_ program can be either an application or an applet.
* FORTRAN
* C++
* C
* **Java**
* LISP and Scheme are both \_\_\_\_\_\_\_ languages.
* procedural
* **functional**
* declarative
* object-oriented
* Prolog is an example of a(n) \_\_\_\_\_\_\_ language.
* procedural
* functional
* **declarative**
* object-oriented
* One phase in system development is \_\_\_\_\_\_\_.
* **analysis**
* application
* designing
* collecting
* Defining the users, requirements, and methods is part of the \_\_\_\_\_\_\_ phase.
* analysis
* **design**
* implementation
* testing
* In the system development process, writing the program is part of the \_\_\_\_\_\_\_ phase.
* analysis
* design
* **implementation**
* testing
* In the system development process, structure charts are tools used in the \_\_\_\_\_\_\_ phase.
* analysis
* **design**
* implementation
* testing
* Testing a software system can involve \_\_\_\_\_\_\_ testing.
* black-box
* glass-box
* neither black-box nor glass-box
* **both black-box and glass-box**
* \_\_\_\_\_\_\_ is the breaking up of a large project into smaller parts.
* Coupling
* Incrementing
* Obsolescence
* **Modularization**
* \_\_\_\_\_\_\_ is a measure of how tightly two modules are bound to each other.
* Modularity
* **Coupling**
* Interoperability
* Cohesion
* \_\_\_\_\_\_\_\_\_between modules in a software system must be minimized.
* **Coupling**
* Cohesion
* Neither coupling nor cohesion
* Both coupling and cohesion
* \_\_\_\_\_\_\_\_\_between modules in a software system must be maximized.
* Coupling
* **Cohesion**
* Neither coupling nor cohesion
* Both coupling and cohesion

Quizzes: Chapter 11

* A data structure can be \_\_\_\_\_\_\_.
* only an array
* only a record
* only a linked list
* **an array, a record, or a linked list**
* An array that consists of just rows and columns is a \_\_\_\_\_\_\_ array.
* one-dimensional
* **two-dimensional**
* three-dimensional
* multidimensional
* Each element in a record is called \_\_\_\_\_\_\_.
* a variable
* an index
* **a field**
* a node
* All the members of a record must be \_\_\_\_\_\_\_.
* the same type
* **related types**
* integer type
* character type
* \_\_\_\_\_\_\_ is an ordered collection of data in which each element contains the location of the next element.
* An array
* A record
* **A linked list**
* A file
* In a linked list, each element contains \_\_\_\_\_\_\_.
* only data
* only a link
* neither data nor a link
* **data and a link**
* The \_\_\_\_\_\_\_ is a pointer that identifies the next element in the linked list.
* **link**
* node
* array
* data
* Given a linked list called *children,* the pointer variable *children* identifies \_\_\_\_\_\_\_\_ element of the linked list.
* **the first**
* the second
* the last
* any
* An empty linked list consists of \_\_\_\_\_\_\_.
* a node
* two nodes
* data and a link
* **a null head pointer**
* To traverse a list, you need a \_\_\_\_\_\_\_ pointer.
* null
* walking
* **beginning**
* insertion

Quizzes: Chapter 12

* In an abstract data type, \_\_\_\_\_\_\_.
* the ADT implementation is known
* **the ADT implementation is hidden**
* the ADT public operations are hidden
* Nothing is hidden
* A stack is a \_\_\_\_\_\_\_\_\_ structure.
* FIFO
* **LIFO**
* DIFO
* SIFO
* A(n) \_\_\_\_\_\_\_ list is also known as a queue.
* LIFO
* **FIFO**
* unordered
* ordered
* If A is the first data element input into a stack, followed by B, C, and D, then \_\_\_\_\_\_\_ is the first element to be removed.
* A
* B
* C
* **D**
* If A is the first data element input into a queue, followed by B, C, and D, then \_\_\_\_\_\_\_ is the first element to be removed.
* **A**
* B
* C
* D
* The pop operation \_\_\_\_\_\_\_ of the stack.
* deletes an item from the top
* deletes an item from the bottom
* **inserts an item at the top**
* inserts an item at the bottom
* The push operation \_\_\_\_\_\_\_ of the stack.
* deletes an item from the top
* deletes an item from the bottom
* **inserts an item at the top**
* inserts an item at the bottom
* In a binary tree, each node has \_\_\_\_\_\_\_ two subtrees.
* more than
* less than
* **at most**
* at least
* In preorder traversal of a binary tree, the \_\_\_\_\_\_.
* left subtree
* is processed first
* right subtree is processed first
* **root is processed first**
* the root is never processed
* In \_\_\_\_\_\_\_ traversal of a binary tree, the right subtree is processed last.
* preorder
* inorder
* **postorder**
* any order
* In postorder traversal of a binary tree, the root is processed \_\_\_\_\_\_\_.
* first
* second
* **last**
* after the left subtree
* In postorder traversal of a binary tree, the left subtree is processed \_\_\_\_\_\_\_.
* first
* second
* last
* **after the right subtree**
* In \_\_\_\_\_\_\_ traversal of a binary tree, the left subtree is processed last.
* preorder
* **inorder**
* postorder
* out of order
* In an inorder traversal of a binary tree, the root is processed \_\_\_\_\_\_\_.
* **first**
* second
* last
* two times

Quizzes: Chapter 13

* \_\_\_\_\_\_\_ file can be accessed randomly.
* A sequential
* **An indexed**
* A hashed
* Any
* \_\_\_\_\_\_\_ file can be accessed sequentially.
* **A sequential**
* An indexed
* A hashed
* No
* When a sequential file is updated, the \_\_\_\_\_\_ file gets the actual update.
* new master
* **old master**
* transaction
* error report
* When a sequential file is updated, the \_\_\_\_\_\_ file contains a list of all errors occurring during the update process.
* new master
* old master
* transaction
* **error report**
* When a sequential file is updated, the \_\_\_\_\_\_ file contains the changes to be applied.
* new master
* old master
* **transaction**
* error report
* After a sequential file is updated, the \_\_\_\_\_\_\_ file contains the most current data.
* **new master**
* old master
* transaction
* error report
* If the transaction file key is 20 and the first master file key is 25, then we \_\_\_\_\_\_\_.
* add the new record to the new master file
* revise the contents of the old master file
* delete the data
* **write the old master file record to the new master file**
* If the transaction file key is 20 with a delete code and the master file key is 20, then we \_\_\_\_\_\_\_.
* add the transaction to the new master file
* revise the contents of the old master file
* **delete the data**
* write the old master file record to the new master file
* An indexed file consists of \_\_\_\_\_\_\_.
* only a sequential data file
* only an index
* only a random data file
* **an index**
* and random data file
* The index of an indexed file has \_\_\_\_\_\_\_ fields.
* **two**
* three
* four
* any number of
* In the \_\_\_\_\_\_\_ hashing method, selected digits are extracted from the key and used as the address.
* direct
* division remainder
* modulo division
* **digit extraction**
* In the \_\_\_\_\_\_\_ hashing method, the key is divided by the file size, and the address is the remainder plus 1.
* direct
* modulo division
* **division remainder**
* digit extraction
* In the \_\_\_\_\_\_\_ hashing method, there are no synonyms or collisions.
* **direct**
* modulo division
* division remainder
* digit extraction
* \_\_\_\_\_\_\_ are keys that hash to the same location in the data file.
* **Collisions**
* Buckets
* Synonyms
* Linked lists
* When a hashing algorithm produces an address for an insertion key and that address is already occupied, it is called a \_\_\_\_\_\_\_.
* **collision**
* probe
* synonym
* linked list
* The address produced by a hashing algorithm is the \_\_\_\_\_\_\_ address.
* probe
* synonym
* collision
* **home**
* The \_\_\_\_\_\_\_ area is the file area that contains all the home addresses.
* probe
* linked
* **hash**
* prime
* In the \_\_\_\_\_\_ collision resolution method, we try to put data that cannot be placed in location 123 into location 124.
* **open addressing**
* linked list
* bucket hashing
* random hashing