If a node is not the first node in a linked list, deleting it may require setting the successor pointer in its predecessor.

|  |  |
| --- | --- |
|  | True |
|  | False |

Moving through a linked list is referred to as \_\_\_\_\_\_\_\_ the list.

|  |  |
| --- | --- |
|  | alternaitng |
|  | cruising |

|  |  |
| --- | --- |
|  | traversing |
|  | node-hopping |

|  |  |
| --- | --- |
|  | None of the above |

A new node must always be made the last node in the list.

|  |  |
| --- | --- |
|  | True |
|  | False |

When an item stored in a linked list is removed, all list items stored after it have to be moved down to plug up the hole.

|  |  |
| --- | --- |
|  | True |
|  | False |

The defining characteristic of a linked list is that

|  |  |
| --- | --- |
|  | the locations that store list data do not have to be consecutive in memory. |
|  | the maximum size of a list is fixed when the list is created. |

|  |  |
| --- | --- |
|  | data are stored in consecutive locations in the memory of the computer. |
|  | lists are very efficient at storing data. |

|  |  |
| --- | --- |
|  | None of the above |

The \_\_\_\_\_\_\_\_ of a linked list points to the first node in the list.

|  |  |
| --- | --- |
|  | tail |
|  | starter |

|  |  |
| --- | --- |
|  | declaration |
|  | head |

|  |  |
| --- | --- |
|  | None of the above |

Insertion into a linked list takes the same number of operations no matter where the insertion occurs

|  |  |
| --- | --- |
|  | True |
|  | False |

In many recursive operations on linked lists,

|  |  |
| --- | --- |
|  | the base case considers the last element of the list. WRONG |
|  | the head of the list is chopped off and thrown away. |

|  |  |
| --- | --- |
|  | the base case is when the list is empty or has a single element. |
|  | All of the above WRONG |

|  |  |
| --- | --- |
|  | None of the above |

To concatenate two linked lists, it is necessary to

|  |  |
| --- | --- |
|  | traverse both lists to get to their ends. |
|  | traverse one of the lists to get to its end. |

|  |  |
| --- | --- |
|  | first reverse both lists. |
|  | first reverse one of the lists. |

|  |  |
| --- | --- |
|  | None of the above. |

In a non-empty list, there must be exactly one list item with no successor.

|  |  |
| --- | --- |
|  | True |
|  | False |

A(n) \_\_\_\_\_\_\_\_ is an abstract data type that stores and retrieves items in a last-in-first-out manner.

|  |  |
| --- | --- |
|  | array |
|  | vector |

|  |  |
| --- | --- |
|  | stack |
|  | queue |

|  |  |
| --- | --- |
|  | None of the above. |

The statement stack< int, vector<int> > iStack; creates

|  |  |
| --- | --- |
|  | a new stack of integers, implemented as a vector. |
|  | a new stack called vector, implemented as integers. |

|  |  |
| --- | --- |
|  | a new vector called stack, implemented with integers. |
|  | a new stack of integers, implemented as a deque. |

|  |  |
| --- | --- |
|  | None of the above. |

The queue data structure is commonly applied in connection with

|  |  |
| --- | --- |
|  | operating systems. |
|  | communications software. |

|  |  |
| --- | --- |
|  | managing the order offprint jobs. |
|  | All the above. |

|  |  |
| --- | --- |
|  | None of the above. |

Two primary queue operations are

|  |  |
| --- | --- |
|  | insert and delete. |
|  | push and pop. |

|  |  |
| --- | --- |
|  | enqueue and dequeue. |
|  | onqueue and offqueue. |

|  |  |
| --- | --- |
|  | None of the above. |

A real-world example of the queue data structure can be seen in a stack of cafeteria trays, where the last tray pushed onto the stack is the first tray removed.

|  |  |
| --- | --- |
|  | True |
|  | False |

A \_\_\_\_\_\_\_\_ is a container that provides quick access to elements at the front and the back of the list.

|  |  |
| --- | --- |
|  | deque |
|  | Stack |

|  |  |
| --- | --- |
|  | queue |
|  | All the above. |

A \_\_\_\_\_\_\_\_ is a double-ended queue.

|  |  |
| --- | --- |
|  | deque |
|  | two-=headed queue |

|  |  |
| --- | --- |
|  | circular array |
|  | twp-tailed vector |

|  |  |
| --- | --- |
|  | None of the above. |

For a stack, a first in last out structure, a doubly linked list would be required.

|  |  |
| --- | --- |
|  | True |
|  | False |

Data is removed from a stack in the \_\_\_\_\_ \_\_\_\_\_\_\_ it was entered.

|  |  |
| --- | --- |
|  | same order |
|  | reverse order |

|  |  |
| --- | --- |
|  | alternating order |
|  | sorted order |

A non-empty linked list of items can be reversed by removing the head, reversing what is left, and then adding the (original) head at the end of the reversed tail.

|  |  |
| --- | --- |
|  | True |
|  | False |

To insert a node into a doubly linked list requires references to the node before and after the location we wish to insert the new node.

|  |  |
| --- | --- |
|  | True |
|  | False |

When you create a linked list, you must know in advance how many nodes the list will contain.

|  |  |
| --- | --- |
|  | True |
|  | False |

Deleting an entire list requires traversing the list to delete the nodes.

|  |  |
| --- | --- |
|  | True |
|  | False |

You enter a series of integer values into a stack and then read them out.  What will the output be?

|  |  |
| --- | --- |
|  | The series of output integers will be in the same order. |
|  | The series of output integers will be in reversed order. |

|  |  |
| --- | --- |
|  | An error as you can only use characters in a stack. |
|  | None of the above. |

A dynamic stack starts as an empty linked list.

|  |  |
| --- | --- |
|  | True |
|  | False |

The first item placed onto a stack is always the last item removed from the stack.

|  |  |
| --- | --- |
|  | True |
|  | False |

A static queue can be implemented as a

|  |  |
| --- | --- |
|  | dynamic linked list. |
|  | stack. |

|  |  |
| --- | --- |
|  | circular array. |
|  | dynamic vector. |

|  |  |
| --- | --- |
|  | None of the above |

The Standard Template Library offers a stack template that may be implemented as a

|  |  |
| --- | --- |
|  | deque. |
|  | vector. |

|  |  |
| --- | --- |
|  | linked list. |
|  | All the above. |

|  |  |
| --- | --- |
|  | None of the above. |

If the head pointer points to NULL, it is an indication that

|  |  |
| --- | --- |
|  | the list is full and cannot accept any new nodes. |
|  | the list has been destroyed. |

|  |  |
| --- | --- |
|  | there are no nodes in the list. |
|  | the list needs to be destroyed. |

|  |  |
| --- | --- |
|  | None of the above |

For large lists, deleting an item from a linked list compared to deletion from an array is

|  |  |
| --- | --- |
|  | more efficient. |
|  | less efficient. |

|  |  |
| --- | --- |
|  | about the same. |
|  | depends on the size of the two lists |

The pop function in the stack template of the STL does not return the value from the top of the stack.

|  |  |
| --- | --- |
|  | True |
|  | False |

For a queue, a first in first out structure, a doubly linked list would be appropriate.

|  |  |
| --- | --- |
|  | True |
|  | False |

Enqueue and dequeue are the two most common stack operations.

|  |  |
| --- | --- |
|  | True |
|  | False |

Inserting an item into a linked list requires that all the items past the point of the insertion be shifted to make room for the new item.

|  |  |
| --- | --- |
|  | True |
|  | False |

For most people, \_\_\_\_\_\_\_\_ queues are more intuitive and easier to understand than \_\_\_\_\_\_\_\_ queues.

|  |  |
| --- | --- |
|  | Static, dynamic WRONG |
|  | Dynamic, static |

|  |  |
| --- | --- |
|  | Stack-like, deque-like |
|  | Deque-like, stack-like |

|  |  |
| --- | --- |
|  | None of the above. |

The \_\_\_\_\_\_\_\_ operation allows an item to be stored on a stack.

|  |  |
| --- | --- |
|  | append |
|  | push |

|  |  |
| --- | --- |
|  | pop |
|  | add |

|  |  |
| --- | --- |
|  | None of the above. |