**A+ Computer Science  
Linked List – Hash Tables  
 M/C TEST**

**Directions ::** On your answer sheet, mark the letter of the best answer to each question.

1. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [] |
| b. | [1] |
| c. | [1, aplus] |
| d. | [aplus, 1] |
| e. | [aplus] |

2. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

System.out.println( bob.size() );

|  |  |
| --- | --- |
| a. | [] |
| b. | 0 |
| c. | null |
| d. | -1 |
| e. | bob |

3. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.addFirst( "1" );

bob.add( 0,"aplus" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [] |
| b. | [1] |
| c. | [1, aplus] |
| d. | [aplus, 1] |
| e. | [aplus] |

4. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.add( "aplus" );

bob.addFirst( "1" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [] |
| b. | [1] |
| c. | [1, aplus] |
| d. | [aplus, 1] |
| e. | [aplus] |

5. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.add( "aplus" );

bob.addFirst( "1" );

bob.addFirst( "comp" );

bob.add( 0, "7" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [7, comp, 1, aplus] |
| b. | [7, 1, comp, aplus] |
| c. | [1, 7, comp, aplus] |
| d. | [aplus, 1, comp, 7] |
| e. | [aplus, comp, 1, 7] |

6. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.add( "aplus" );

bob.addFirst( "1" );

bob.addLast( "comp" );

bob.add( 1, "7" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [7, comp, 1, aplus] |
| b. | [7, 1, comp, aplus] |
| c. | [1, 7, aplus, comp] |
| d. | [aplus, 1, comp, 7] |
| e. | [aplus, comp, 1, 7] |

7. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.add( "d" );

bob.addFirst( "1" );

bob.addLast( "4" );

bob.add( "w" );

bob.addFirst( "0" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [1, 0, d, 4, w] |
| b. | [0, 4, d, 1, w] |
| c. | [0, 1, d, 4, w] |
| d. | [d, 1, 0, 4, w] |
| e. | [0, 1, w, 4, d] |

8. **What is output by the code below?**

LinkedList<String> bob;

bob = new LinkedList<String>();

bob.add( "d" );

bob.addFirst( "1" );

bob.addFirst( "4" );

bob.add( 0, "w" );

bob.set( 1, "x" );

bob.addLast( "0" );

System.out.println(bob);

|  |  |
| --- | --- |
| a. | [w, x, 1, d, 0] |
| b. | [w, 4, d, 1, w] |
| c. | [w, x, d, 4, w] |
| d. | [w, 1, 0, 4, x] |
| e. | [w, x, 1, 4, d] |

9. **What is output by the code below?**

LinkedList<String> ann;

ann = new LinkedList<String>();

ann.add("3");

ann.add("0");

ann.add("2");

ann.add("1");

Collections.sort(ann);

System.out.println(ann);

|  |  |
| --- | --- |
| a. | [0, 1, 2, 3] |
| b. | [3, 2, 1, 0] |
| c. | [3, 0, 2, 1] |
| d. | [1, 2, 0, 3] |
| e. | [0, 1, 2, 1] |

10. **What is output by the code below?**

LinkedList<String> ann;

ann = new LinkedList<String>();

ann.add("3");

ann.add(0,"0");

ann.add("2");

ann.add(0,"1");

System.out.println(ann);

|  |  |
| --- | --- |
| a. | [0, 1, 2, 3] |
| b. | [3, 2, 1, 0] |
| c. | [3, 0, 2, 1] |
| d. | [1, 2, 0, 3] |
| e. | [1, 0, 3, 2] |

11. **What is output by the code below?**

LinkedList<String> ann;

ann = new LinkedList<String>();

ann.add("3");

ann.add(0,"0");

ann.add("2");

ListIterator x = ann.listIterator();

x.next();

x.add( "dog" );

System.out.println(ann);

|  |  |
| --- | --- |
| a. | [0, dog, 3, 2] |
| b. | [0, 3, 2, dog] |
| c. | [dog, 0, 3, 2] |
| d. | [0, dog, 3, 2, dog] |
| e. | [dog] |

12. **What is output by the code below?**

LinkedList<String> ann;

ann = new LinkedList<String>();

ann.add("3");

ann.add(0,"0");

ann.add("2");

ListIterator x = ann.listIterator();

x.next();

x.add( "dog" );

x.add( "cat" );

System.out.println(ann);

|  |  |
| --- | --- |
| a. | [0, dog, 3, 2] |
| b. | [0, 3, 2, dog] |
| c. | [dog, 0, 3, 2] |
| d. | [0, dog, cat, 3, 2] |
| e. | There is no output due to a runtime exception. |

13. **Which of the following blocks of code would print out every single node in a linked list?**

I.

ListNode x = //first node

while( x.getNext()!=null&&x!=null )

{

out.println(x.getValue());

x=x.getNext();

}

II.

ListNode x = //first node

while( x!=null )

{

out.println(x.getValue());

}

III.

ListNode x = //first node

while( x!=null )

{

out.println(x.getValue());

x=x.getNext();

}

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and III only |
| d. | II and III only |
| e. | III only |

14. **Assuming the linked list stores only ints, which of the following blocks of code would sum up every value in every single node?**

I.

int total=0;

ListNode x = //first node

while(x!=null)

{

total+=x.getValue();

x=x.getNext();

}

II.

int total=0;

ListNode x = //first node

while(x.getNext()!=null)

{

total+=x.getValue();

x=x.getNext();

}

III.

int total=0;

ListNode x = //first node

while(x!=null)

{

x=x.getNext();

total+=x.getValue();

}

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and III only |
| d. | II and III only |
| e. | III only |

15. **Which of the following would fill** /\* code 1 \*/  **in the code below to complete method countNodes()?**

//this method will return the count

//of all nodes in the list

public static int countNodes(ListNode list)

{

int count=0;

while( /\* code 1 \*/ )

{

count++;

/\* code 2 \*/

}

return count;

}

|  |  |
| --- | --- |
| a. | list != null |
| b. | list.getNext() != null |
| c. | list == null |
| d. | list = list.next |
| e. | list=list.getNext(); |

16. **Assuming** /\* code 1 \*/  **is filled correctly, which of the following would fill** /\* code 2 \*/  **in the code below to complete method countNodes()?**

//this method will return the count

//of all nodes in the list

public static int countNodes(ListNode list)

{

int count=0;

while( /\* code 1 \*/ )

{

count++;

/\* code 2 \*/

}

return count;

}

|  |  |
| --- | --- |
| a. | list.getNext(); |
| b. | list.setNext(list.getNext()); |
| c. | list=list.getValue(); |
| d. | list.setNext(list.getNext()); |
| e. | list=list.getNext(); |

17. **Which of the following would fill** /\* code 1 \*/ **in the code below to complete method countNodes()?**

//this method will return the count

//of all nodes in the list

public static int countNodes(ListNode list)

{

if( /\* code 1 \*/ )

{

return 0;

}

else

/\* code 2 \*/

}

|  |  |
| --- | --- |
| a. | list != null |
| b. | list == null |
| c. | list.getNext() != null |
| d. | list.next != null |
| e. | list |

18. **Assuming** /\* code 1 \*/ **was filled correctly, which of the following would fill** /\* code 2 \*/ **in the code below to complete method countNodes()?**

//this method will return the count

//of all nodes in the list

public static int countNodes(ListNode list)

{

if( /\* code 1 \*/ )

{

return 0;

}

else

/\* code 2 \*/

}

|  |  |
| --- | --- |
| a. | return 1 + countNodes(list.getNext()); |
| b. | return countNodes(list.getNext()); |
| c. | return 1 + countNodes(list); |
| d. | return 1 + list.getNext(); |
| e. | return 0 + countNodes(list.getNext()); |

19. **What of the following would fill // line 1 in the code at right to complete method doubleFront()?**

//this method will add a new node to the

//list with the same value as the current

//node at the front of the list

public static void doubleFront(ListNode list)

{

ListNode add;

add = new ListNode(list.getValue(),

list.getNext());

// line 1

}

|  |  |
| --- | --- |
| a. | list=add; |
| b. | list.setNext(add); |
| c. | list=list.getNext(); |
| d. | list.setNext(list.getNext()); |
| e. | list=add.getNext(); |

20. **Give the class below, which method would most likely contain the code that follows?**

public class LinkedList

{

private ListNode front, back;

//methods

}

//code for which method?

if(front==null)

front = back = new ListNode(obj,front);

else

front = new ListNode(obj,front);

|  |  |
| --- | --- |
| a. | addFirst(Object obj) |
| b. | removeFirst() |
| c. | addLast(Object obj) |
| d. | removeLast() |
| e. | remove() |

21. **Give the class below, which method would most likely contain the code that follows?**

public class LinkedList

{

private ListNode front, back;

//methods

}

//code for which method?

if(front==null)

front = back = new ListNode(obj,front);

else{

ListNode temp = new ListNode(obj,null);

back.setNext(temp);

back = temp;

}

|  |  |
| --- | --- |
| a. | addFirst(Object obj) |
| b. | removeFirst() |
| c. | addLast(Object obj) |
| d. | removeLast() |
| e. | none of these |

22. **Give the class below, which method would most likely contain the code that follows?**

public class LinkedList

{

private ListNode front, back;

//methods

}

//code for which method?

Object obj = new Object();

if(front!=null)

{

ListNode temp = front;

obj = front.getValue();

front = front.getNext();

temp = null;

}

return obj;

|  |  |
| --- | --- |
| a. | addFirst(Object obj) |
| b. | removeFirst() |
| c. | addLast(Object obj) |
| d. | removeLast() |
| e. | remove() |

23. **Given an instance variable theList that refers to the front of a Linked List, which of the following code segments would add a new node at the front of the Linked List that contains the exact same value as the current node at the front?**

I.

ListNode add = new ListNode(theList.getValue(),null);

theList = add;

II.

theList = new ListNode(theList.getValue(),theList);

III.

ListNode add = new ListNode(theList.getValue(),theList);

theList = add;

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and II only |
| d. | II and III only |
| e. | III only |

24. **Given an instance variable theList that refers to the front of a Linked List, which of the following code segments would add a new node at the end of the Linked List that contains the exact same value as the current node at the end?**

I.

ListNode list = theList;

while(list!=null&&list.getNext()!=null){

list=list.getNext();

}

list.setNext(new ListNode(list.getValue(),null));

II.

ListNode list = theList;

while(list!=null){

list=list.getNext();

}

list.setNext(new ListNode(list.getValue(),null));

III.

ListNode list = theList;

while(list!=null&&list.getNext()!=null){

list=list.getNext();

}

list.setNext(new ListNode(list.getValue(),list));

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and II only |
| d. | II and III only |
| e. | III only |

25. **Given an instance variable theList that refers to the front of a Linked List, which of the following would count up the number of nodes in the Linked List without modifying the Linked List?**

I.

ListNode list = theList;

int count=0;

while(list!=null&&list.getNext()!=null){

count++;

list=list.getNext();

}

II.

int count=0;

while(theList!=null){

count++;

theList=theList.getNext();

}

III.

ListNode list = theList;

int count=0;

while(list!=null){

count++;

list=list.getNext();

}

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and II only |
| d. | II and III only |
| e. | III only |

26. **Given a variable theList that refers to the front of a Linked List, which of the following code segments would add a new node at the front of the Linked List that contains the word "hello".**

I.

ListNode add = new ListNode("hello",theList);

theList = add;

II.

theList = new ListNode("hello",theList);

III.

theList = new ListNode("hello",null);

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and II only |
| d. | II and III only |
| e. | III only |

27. **After running the code below, how many values would the linked list at spot 0 contain?**

public class HashTable

{

private int size;

private ListNode[] table;

public HashTable(int numSlots) {

size = numSlots;

table = new ListNode[size];

}

public void add(Object obj)

{

ListNode element = new ListNode(obj, null);

int index = obj.hashCode() % size;

if(table[index] == null)

{

table[index] = element;

}

else

{

ListNode current = table[index];

while(current.getNext() != null)

current = current.getNext();

current.setNext(element);

}

}

}

//client code

HashTable h = new HashTable(10);

int[] nums = {10,7,8,13,11,4,5,1,12,5,6,9,2,3,17,27,31,35};

for( int val : nums )

{

h.add( val );

}

|  |  |
| --- | --- |
| a. | 0 |
| b. | 1 |
| c. | 2 |
| d. | 3 |
| e. | 4 |

28. **After running the code below, how many values would the linked list at spot 3 contain?**

public class HashTable

{

private int size;

private ListNode[] table;

public HashTable(int numSlots) {

size = numSlots;

table = new ListNode[size];

}

public void add(Object obj)

{

ListNode element = new ListNode(obj, null);

int index = obj.hashCode() % size;

if(table[index] == null)

{

table[index] = element;

}

else

{

ListNode current = table[index];

while(current.getNext() != null)

current = current.getNext();

current.setNext(element);

}

}

}

//client code

HashTable h = new HashTable(10);

int[] nums = {10,7,8,13,11,4,5,1,12,5,6,9,2,3,17,27,31,35};

for( int val : nums )

{

h.add( val );

}

|  |  |
| --- | --- |
| a. | 0 |
| b. | 1 |
| c. | 2 |
| d. | 3 |
| e. | 4 |

29. **After running the code below, how many values would the linked list at spot 7 contain?**

public class HashTable

{

private int size;

private ListNode[] table;

public HashTable(int numSlots) {

size = numSlots;

table = new ListNode[size];

}

public void add(Object obj)

{

ListNode element = new ListNode(obj, null);

int index = obj.hashCode() % size;

if(table[index] == null)

{

table[index] = element;

}

else

{

ListNode current = table[index];

while(current.getNext() != null)

current = current.getNext();

current.setNext(element);

}

}

}

//client code

HashTable h = new HashTable(10);

int[] nums = {10,7,8,13,11,4,5,1,12,5,6,9,2,3,17,27,31,35};

for( int val : nums )

{

h.add( val );

}

|  |  |
| --- | --- |
| a. | 0 |
| b. | 1 |
| c. | 2 |
| d. | 3 |
| e. | 4 |

30. **After running the code below, how many values would the linked list at spot 9 contain?**

public class HashTable

{

private int size;

private ListNode[] table;

public HashTable(int numSlots) {

size = numSlots;

table = new ListNode[size];

}

public void add(Object obj)

{

ListNode element = new ListNode(obj, null);

int index = obj.hashCode() % size;

if(table[index] == null)

{

table[index] = element;

}

else

{

ListNode current = table[index];

while(current.getNext() != null)

current = current.getNext();

current.setNext(element);

}

}

}

//client code

HashTable h = new HashTable(10);

int[] nums = {10,7,8,13,11,4,5,1,12,5,6,9,2,3,17,27,31,35};

for( int val : nums )

{

h.add( val );

}

|  |  |
| --- | --- |
| a. | 0 |
| b. | 1 |
| c. | 2 |
| d. | 3 |
| e. | 4 |

31. Which of the following best matches the runtime for the following LinkedList functions?

addFirst() getFirst()

LINE 1 O(1) O(1)

LINE 2 O(1) O(N)

LINE 3 O(N) O(N)

LINE 4 O(log2N) O(log2N)

LINE 5 O(log2N) O(N)

|  |  |
| --- | --- |
| a. | Line 1 |
| b. | Line 2 |
| c. | Line 3 |
| d. | Line 4 |
| e. | Line 5 |

32. Which of the following best matches the runtime for the following LinkedList functions?

public E get(int index)

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N2) |
| d. | O(log2N) |
| e. | O(N!) |

33. When referring to hash functions what is a collision?

|  |  |
| --- | --- |
| a. | When a single item has multiple different hashes |
| b. | The slot in the hash table an item hashes to |
| c. | When a hash table runs out of space |
| d. | When two keys hash to the same slot |
| e. | When duplicate items hash to different slots |

34. How does chaining or using buckets resolve collisions?

|  |  |
| --- | --- |
| a. | Waiting until the slot is free to insert into the hash table |
| b. | Adding together multiple results of the same hash function until given a free slot |
| c. | Recursively calling the hash function to get a different hash |
| d. | Resizing the hash table to fit more items |
| e. | Placing the items with the same hash in a linked list |

35. What advantage does a hash table have over a direct-address table?

|  |  |
| --- | --- |
| a. | Faster access to elements |
| b. | Less space wasted on giant sets of keys |
| c. | Elements are sorted |
| d. | Improved memory locality of elements |
| e. | log2N search for any element |

36. What is a sentinel node in linked lists?

|  |  |
| --- | --- |
| a. | A dummy node that specifies the end of the linked list |
| b. | A node that ensures secure access to the list |
| c. | A node that checks the integrity of the values inserted into the list |
| d. | It refers to the previous node in the list |
| e. | It refers to the next node in the list |

37. **Given an instance variable theList that refers to the front of a Linked List, which of the following code segments would remove the first node with a value equal to obj?**

I.

Object obj;

ListNode list = theList, prev = null;

while(list!=null){

if (list.getValue().equals(obj)) {

if (prev != null)

prev.setNext(list.getNext());

else

theList = theList.getNext();

break;

}

prev = list;

list = list.getNext();

}

II.

Object obj;

ListNode list = theList, prev = null;

while(list!=null){

if (list.getValue().equals(obj)) {

if (prev != null)

prev.setNext(list.getNext());

else

theList = theList.getNext();

}

prev = list;

list = list.getNext();

}

III.

Object obj;

ListNode list = theList, prev = null;

while(list!=null){

if (list.getValue().equals(obj)) {

if (prev != null)

prev.setNext(list.getNext());

else

theList = theList.getNext();

break;

}

}

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | I and II only |
| d. | II and III only |
| e. | III only |

38. What is the difference between a singly-linked list and a doubly-linked list?

|  |  |
| --- | --- |
| a. | A doubly-linked list has a copy of each value in the list |
| b. | A doubly-linked list has two next pointers that act like child nodes |
| c. | A doubly-linked list contains both a previous and next pointer |
| d. | A singly-linked list has no duplicates |
| e. | A singly-linked list takes half the number of bits to represent |

39. Unlike an array where the order of elements is determined by the position in memory, a linked list’s ordering is based on what?

|  |  |
| --- | --- |
| a. | The hash functions of the elements |
| b. | An index value that each node contains |
| c. | The amount of memory the elements occupy |
| d. | The pointers between nodes in the linked list |
| e. | The natural ordering of the elements |

40. Which of the following best matches the running time of accessing an element in these two lists?

array LinkedList

LINE 1 O(1) O(1)

LINE 2 O(1) O(N)

LINE 3 O(1) O(N2)

LINE 4 O(N) O(N)

LINE 5 O(N) O(N2)

|  |  |
| --- | --- |
| a. | Line 1 |
| b. | Line 2 |
| c. | Line 3 |
| d. | Line 4 |
| e. | Line 5 |