|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in IT, ISF  Year 2 (2017/2018) Semester 4 | **Week 3** |
| **1-2 Hours** |
| **Tutorial 3 – Linked List** | |

1. Explain the purpose of the following operators

1. &

|  |
| --- |
| The ampersand operator to retrieve the address of the variable. |

1. \*

|  |
| --- |
| The dereference operator allows a pointer variable to point to a value.  //Correct Answer: To retrieve the value pointed to by a pointer variable |

1. new

|  |
| --- |
| The new keyword allows an instantiation of an object/struct  //Correct answer: to create a data object dynamically, at run-time. |

2. Assuming the Node structure is declared as:

struct Node

{

string item; // to store the data item

Node \*next; // pointer to point to next node

};

1. Draw the diagrams to show what happen in the computer memory when the following statements are executed.

Node node1;

Node \*node2;

|  |
| --- |
| Node node1;  item  NULL |
| Node \*node2; |

1. Write the statements to:

(i) store “Kevin” in node1.

(ii) initialize the next pointer in node1 to NULL

|  |
| --- |
| 1. Node1.item = “Kevin”; 2. Node1.next = NULL; |

1. Write the statements to:
2. create a Node object and set node2 to point to it
3. store “Vivian” in Node object, pointed to by node2.

(iii) initialize the next pointer in Node object, pointed to by node2, to NULL

|  |
| --- |
| 1. Node2 = new Node; 2. Node2 ->item = “Vivian”; 3. Node2->next = NULL; |

1. What happen when the following statements are executed?

Node node3 = node1;

|  |
| --- |
| Node3 will have a duplicate copy of node1.  **Changes in node3 will not affect node1.** |

Node \*node4 = node2;

|  |
| --- |
| Both node4 and node2 will point to the same Node object.  //need to relook this. |

3. The specification of the List ADT implemented using Pointers is given below.

|  |
| --- |
| *// List.h - - Specification of List ADT (implemented using Pointers)*  #pragma once  #include<string>  #include<iostream>  using namespace std;  typedef string ItemType;  class List  {  private:  struct Node  {  ItemType item; // item  Node \*next; // pointer pointing to next item  };  Node \*firstNode; // point to the first item  int size; // number of items in the list  public:  // constructor  List();  *// add an item to the back of the list (append)*  bool add(ItemType item);  *// add an item at a specified position in the list (insert)*  bool add(int index, ItemType item);  *// remove an item at a specified position in the list*  void remove(int index);  *// get an item at a specified position of the list (retrieve)*  ItemType get(int index);  *// check if the list is empty*  bool isEmpty();  *// check the size of the list*  int getLength();  }; |

Implement the following operations of the List ADT

(a) bool add(ItemType item)

|  |
| --- |
| {  // create a new node to store data  Node \*newNode = new Node; //this is dynamic.  newNode -> item = item;  newNode->next = NULL;  if (firstNode == NULL)  firstNode = newNode;  else{  Node \*previous = firstNode;  Node\* current = firstNode;  While (current !=NULL){  Previous = current;  Current = current->next;  }  Size++;  Return true;  } |

(b) bool add(int index, ItemType item);

|  |
| --- |
|  |

(c) int getLength()

|  |
| --- |
|  |