|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in IT, ISF, FI  Year 2 (2018/2019) Semester 4 | **Week 3** |
| **1-2 Hours** |
| **Tutorial 3 – Linked List** | |

1. Explain the purpose of the following operators

1. **&**

|  |
| --- |
| Gets the memory address location of a data variable |

1. **\***

|  |
| --- |
| Gets the value at the memory address pointed by the pointer |

1. **new**

|  |
| --- |
| Creates a new object dynamically at runtime. |

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;   |  |  |  |  | | --- | --- | --- | --- | | Value | node1  item: dummy string  next: dummy pointer to Node |  |  | | Memory address | 0x0001 |  |  | |
| Node \*node2;   |  |  |  |  | | --- | --- | --- | --- | | Value | node1 | 0x0001  dummy pointer to node |  | | Memory address | 0x0001 | 0x0002 |  | |

1. Write the statements to:

(i) store “Kevin” in node1.

(ii) initialize the next pointer in node1 to NULL

|  |
| --- |
| node1.item = “Kevin”  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

|  |
| --- |
| (i)  node2 = new Node;  node2->next = node3; |
| (ii)  node3->item = “Vivian” |
| (iii)  node3->next = NULL |

1. What happen when the following statements are executed?

Node node3 = node1;

|  |
| --- |
| node3 is a copy of node1 – changes in node3 won’t affect node1. |

Node \*node4 = node2;

|  |
| --- |
| \*node4 is pointing towards node2 |

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)

|  |
| --- |
| bool add(ItemType item)  {  Node \*newNode = new Node;  newNode->item = i;  newNode->next = NULL;  if (size == 0) //or if (firstNode->next == NULL)  {  firstNode = newNode;  }  else  {  Node \*temp = new Node;  temp = firstNode;  for (int i = 1; i < size ; i++)  }  temp = temp->next;  temp->next = newNode;  size++;    } |

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

|  |
| --- |
| bool add(int index, ItemType item)  {  bool success = (index >= 1 & index <= size);  if (success)  {  Node \* newNode = new Node;  newNode->item = item;  newNode->next = NULL;  if (index == 1)  {  newNode->next = firstNode;  firstNode = newNode;  }  else  {  Node \*temp = firstNode;  for (int i = 1; i < index -1; i++)  {  temp = temp->next;  }  newNode->next = temp->next;  temp->next = newNode;  size++;  }  }  } |

(c) int getLength()

|  |
| --- |
| int getLength()  {  return size;  } |