# Comp 4603

# Advanced C++

|  |  |  |  |
| --- | --- | --- | --- |
| Assignment | 2 | Part | 1 |

Student Name: Alisher Shamayev

BCIT ID: A01182685

Task 1: To define a function which takes in an array of integer, find and print both the second largest value and second smallest values.

Eg, input array [3, 4, 5, 2, 1] => to print 4 and 2

|  |
| --- |
| //  // main.cpp  // Lab2P1.1  //  // Created by GUW06 on 2023-01-19.  //  **int** findLargest(){  **int** myList[]={3,4,5,2,1};  **int** max\_Value = myList[0];  **int** secondMax=0;    **for** (**int** i = 1; i < **sizeof**(myList)/**sizeof**(**int**); i++)  {  **if** (myList[i] > max\_Value)  {  secondMax = max\_Value;  max\_Value=myList[i];  }  **else** **if**(myList[i]>secondMax){  secondMax=myList[i];  }    }    **return** secondMax;  }  **int** findSmallest(){  **int** myList[]={3,4,5,2,1};  **int** min\_Value = myList[0];  **int** secondMin=0;  **for** (**int** i = 1; i < **sizeof**(myList)/**sizeof**(**int**); i++)  {  **if** (myList[i]<min\_Value ) {  secondMin=min\_Value;  min\_Value=myList[i];  }  **else** **if**(myList[i]<secondMin){  secondMin=myList[i];  }    }  **return** secondMin;  }  #include <iostream>  **using** **namespace** std;  **int** main(**int** argc, **const** **char** \* argv[]) {  // insert code here...  **int** myList[]={3,4,5,2,1};  cout<<"The array elements are: "<<endl;  **for**(**int** e:myList){  cout<<e<<",";  }    cout<<"\nSecond largest value is: "<<findLargest()<<endl;  cout<<"Second smallest value is: "<<findSmallest()<<endl;    } |

Task 2: To implement listed function:

|  |
| --- |
| **template**<**typename** T>  **void** LinkedList<T>::add(**int** index, T element)  {  **if** (index == 0){  Node<T>\* newNode = **new** Node<T>(element);  newNode->next = head;  head = newNode;  size++;  **if** (tail == **NULL**)  tail = head;  }  **else** **if** (index >= size){  **if** (tail == **NULL**)  {  head = tail = **new** Node<T>(element);  }  **else** {  tail->next = **new** Node<T>(element);  tail = tail->next;  }  size++;    }  **else**  {  Node<T>\* current = head;  **for** (**int** i = 1; i < index; i++)  current = current->next;  Node<T>\* temp = current->next;  current->next = **new** Node<T>(element);  (current->next)->next = temp;  size++;  }  } |

Task 3:To implement listed function

|  |
| --- |
| **template**<**typename** T>  T LinkedList<T>::removeLast() **throw** (runtime\_error)  {  **if** (size == 0){  **throw** runtime\_error("List is empty");  }  **else** **if** (size == 1)  {  Node<T>\* temp = head;  head = tail = **NULL**;  size = 0;  T element = temp->element;  **delete** temp;  **return** element;  }  **else**  {  Node<T>\* current = head;  **for** (**int** i = 0; i < size - 2; i++){  current = current->next;  }  Node<T>\* temp = tail;  tail = current;  tail->next = **NULL**;  size--;  T element = temp->element;  **delete** temp;  **return** element;  }  } |

Task 4: To implement listed function

|  |
| --- |
| **template**<**typename** T>  T LinkedList<T>::removeAt(**int** index) **throw** (runtime\_error)  {  **if** (index < 0 || index >= size){  **throw** runtime\_error("Index is not found, should be between 0 and size of List");  }  **else** **if** (index == 0){  **if** (size == 0){  **throw** runtime\_error("List is empty");  }  **else**  {  Node<T>\* temp = head;  head = head->next;  size--;  **if** (head == **NULL**) tail = **NULL**;  T element = temp->element;  **delete** temp;  **return** element;  }  }  **else** **if** (index == size - 1){  **return** removeLast();  }  **else** {  Node<T>\* previous = head;  **for** (**int** i = 1; i < index; i++)  {  previous = previous->next;  }  Node<T>\* current = previous->next;  previous->next = current->next;  size--;  T element = current->element;  **delete** current;  **return** element;  }  } |