**North South University - Spring 2023**

Course: CSE225L Assessment: Lab 6

Section: 6 NSU ID: 2211424642 Name: Joy Kumar Ghosh

// Task-1 Codes

|  |
| --- |
| #include <iostream>  using namespace std;  //class declaration  template <class ItemType>  class UnsortedType  {  struct NodeType  {  ItemType info;  NodeType\* next;  };  public:  UnsortedType();  ~UnsortedType();  bool IsFull();  int LengthIs();  void MakeEmpty();  void RetrieveItem(ItemType &, bool &);  void InsertItem(ItemType);  void DeleteItem(ItemType);  void ResetList();  void GetNextItem(ItemType&);  private:  NodeType\* listData;  int length;  NodeType\* currentPos;  }; |
| //implementation  template <class ItemType>  UnsortedType<ItemType>::UnsortedType(){  length = 0;  listData = NULL;  currentPos = NULL;  }  template <class ItemType>  int UnsortedType<ItemType>::LengthIs(){  return length;  }  template<class ItemType>  bool UnsortedType<ItemType>::IsFull(){  NodeType\* location;  try  {  location = new NodeType;  delete location;  return false;  }  catch(bad\_alloc& exception)  {  return true;  }  }  template <class ItemType>  void UnsortedType<ItemType>::InsertItem(ItemType item){  NodeType\* location;  location = new NodeType;  location->info = item;  location->next = listData;  listData = location;  length++;  }  template <class ItemType>  void UnsortedType<ItemType>::DeleteItem(ItemType item){  NodeType\* location = listData;  NodeType\* tempLocation;  if (item == listData->info){  tempLocation = location;  listData = listData->next;  }  else{  while (!(item==(location->next)->info))  location = location->next;  tempLocation = location->next;  location->next = (location->next)->next;  }  delete tempLocation;  length--;  }  template <class ItemType>  void UnsortedType<ItemType>::RetrieveItem(ItemType &item, bool &found){  NodeType\* location = listData;  bool moreToSearch = (location != NULL);  found = false;  while (moreToSearch && !found){  if (item == location->info)  found = true;  else{  location = location->next;  moreToSearch = (location != NULL);  }  }  }  template <class ItemType>  void UnsortedType<ItemType>::MakeEmpty(){  NodeType\* tempPtr;  while (listData != NULL){  tempPtr = listData;  listData = listData->next;  delete tempPtr;  }  length = 0;  }  template <class ItemType>  UnsortedType<ItemType>::~UnsortedType(){  MakeEmpty();  }  template <class ItemType>  void UnsortedType<ItemType>::ResetList(){  currentPos = NULL;  }  template <class ItemType>  void UnsortedType<ItemType>::GetNextItem(ItemType &item){  if (currentPos == NULL)  currentPos = listData;  else  currentPos = currentPos->next;  item = currentPos->info;  } |
| //main driver file  int main()  {  UnsortedType<int> integerList;  int item, i;  bool isFound = false;  //inserting 4 item  for(i = 0; i < 4; i++){  cin >> item;  integerList.InsertItem(item);  }  cout << endl;  //printing list  for(i = 0, integerList.ResetList(); i < integerList.LengthIs(); i++){  integerList.GetNextItem(item);  cout << item << " ";  }  cout << endl;  //printing length  cout << integerList.LengthIs() << endl;  //inserting item  cin >> item;  integerList.InsertItem(item);  //printing list  for(i = 0, integerList.ResetList(); i < integerList.LengthIs(); i++){  integerList.GetNextItem(item);  cout << item << " ";  }  cout << endl;  //retrieving 4  item = 4;  integerList.RetrieveItem(item, isFound);  if(isFound)  cout << "Item is Found." << endl;  else  cout << "Item is not Found" << endl;  //retrieving 5  item = 5;  integerList.RetrieveItem(item, isFound);  if(isFound)  cout << "Item is Found." << endl;  else  cout << "Item is not Found" << endl;  //retrieving 9  item = 9;  integerList.RetrieveItem(item, isFound);  if(isFound)  cout << "Item is Found." << endl;  else  cout << "Item is not Found" << endl;  //retrieving 10  item = 10;  integerList.RetrieveItem(item, isFound);  if(isFound)  cout << "Item is Found." << endl;  else  cout << "Item is not Found" << endl;  //printing list full or not  if(integerList.IsFull()){  cout << "List is Full out of Memory.." << endl;  }  else  cout << "List is not full.We have enough memory.." << endl;  //deleting 5  integerList.DeleteItem(5);  //printing list full or not  if(integerList.IsFull()){  cout << "List is Full out of Memory.." << endl;  }  else  cout << "List is not full.We have enough memory.." << endl;  //deleting 1  integerList.DeleteItem(1);  //printing list  for(i = 0, integerList.ResetList(); i < integerList.LengthIs(); i++){  integerList.GetNextItem(item);  cout << item << " ";  }  cout << endl;  //deleting 6  integerList.DeleteItem(6);  //printing list  for(i = 0, integerList.ResetList(); i < integerList.LengthIs(); i++){  integerList.GetNextItem(item);  cout << item << " ";  }  cout << endl;  } |
|  |

// Task-2 Codes

|  |
| --- |
|  |
|  |
|  |

// Task-3 Codes

|  |
| --- |
|  |
|  |
|  |

// Task-4 Codes

|  |
| --- |
|  |
|  |
|  |

// Task-5 Codes

|  |
| --- |
|  |
|  |
|  |