Instructions: Please read carefully

- Please rename this file as only your ID number **(e.g. 18-*****-1.doc or 18-*****-1.pdf).**
- Submit the file before **11:59pm on 04/12/2020** in the Portal Lab Performance section labeled **Lab task 9.** **If you cannot complete the full task, do not worry. Just upload what you have completed.**

1. Do the following to write program for a Single Linked List:

1. Create a singly linked list by **inserting** node one by one at the end.
2. **Display** your list
3. **Insert** a new item at a specific position (at the beginning and after a given node)
4. **Search** an item into your linked list.
5. **Delete** an item from the list (at beginning, at last and at middle)

Take users input from the console to perform the operations.

Your code here:

```
#include<iostream>

using namespace std;

class Node {
  public:
    int key;
  int data;
  Node * next;

  Node() {
    key = 0;
    data = 0;
    next = NULL;
  }
  Node(int k, int d) {
    key = k;
    data = d;
  }
};

class SinglyLinkedList {
  public:
    Node * head;

  SinglyLinkedList() {
    head = NULL;
  }
  SinglyLinkedList(Node * n) {
    head = n;
  }

  // 1. CHeck if node exists using key value
  Node * nodeExists(int k) {
    Node * temp = NULL;

    Node * ptr = head;
    while (ptr != NULL) {
```

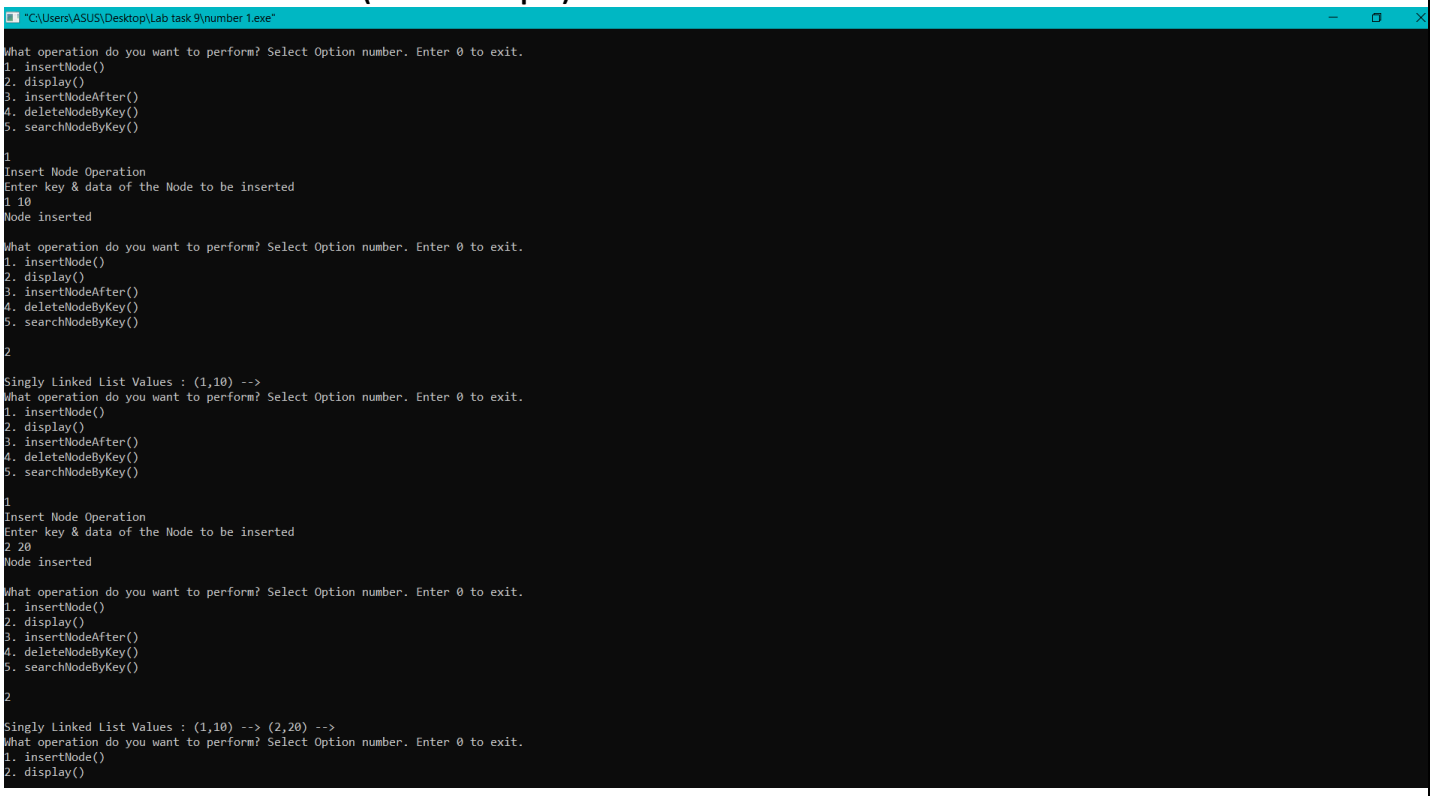
```
    if (ptr -> key == k) {
      temp = ptr;
    }
    ptr = ptr -> next;

  }
  return temp;
 }

 // 2. Insert a node to the list
 void insertNode(Node * n) {
  if (nodeExists(n -> key) != NULL) {
    cout << "Node Already exists with key value : " << n -> key << ". Insert another node with different Key value" <<
endl;
  } else {
    if (head == NULL) {
      head = n;
      cout << "Node inserted" << endl;
    } else {
      Node * ptr = head;
      while (ptr -> next != NULL) {
        ptr = ptr -> next;
      }
      ptr -> next = n;
      cout << "Node inserted" << endl;
    }
  }

 }


  // 3. Display list
 void displayList() {
  if (head == NULL) {
    cout << "No Nodes in Singly Linked List";
  } else {
    cout << endl << "Singly Linked List Values : ";
    Node * temp = head;

    while (temp != NULL) {
      cout << "(" << temp -> key << "," << temp -> data << ") --> ";
      temp = temp -> next;
    }
  }

 }


 // 4. Insert a Node after a given node
 void insertNodeAfter(int k, Node * n) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value: " << k << endl;
```

```
    } else {
      if (nodeExists(n -> key) != NULL) {
        cout << "Node Already exists with key value : " << n -> key << ". Append another node with different Key value"
<< endl;
      } else {
        n -> next = ptr -> next;
        ptr -> next = n;
        cout << "Node Inserted" << endl;
      }
    }
  }

  // 5. Delete node by unique key
  void deleteNodeByKey(int k) {
    if (head == NULL) {
      cout << "Singly Linked List already Empty. Cant delete" << endl;
    } else if (head != NULL) {
      if (head -> key == k) {
        head = head -> next;
        cout << "Node UNLINKED with keys value : " << k << endl;
      } else {
        Node * temp = NULL;
        Node * prevptr = head;
        Node * currentptr = head -> next;
        while (currentptr != NULL) {
          if (currentptr -> key == k) {
            temp = currentptr;
            currentptr = NULL;
          } else {
            prevptr = prevptr -> next;
            currentptr = currentptr -> next;
          }
        }
        if (temp != NULL) {
          prevptr -> next = temp -> next;
          cout << "Node UNLINKED with keys value : " << k << endl;
        } else {
          cout << "Node Doesn't exist with key value : " << k << endl;
        }
      }
    }

  }
  // 6. Search node
  void searchNodeByKey(int k, int d) {

    Node * ptr = nodeExists(k);
    if (ptr != NULL) {
      ptr -> data = d;
      cout << "Node Data search Successfully" << endl;
    } else {
      cout << "Node Doesn't exist with key value : " << k << endl;
    }
```

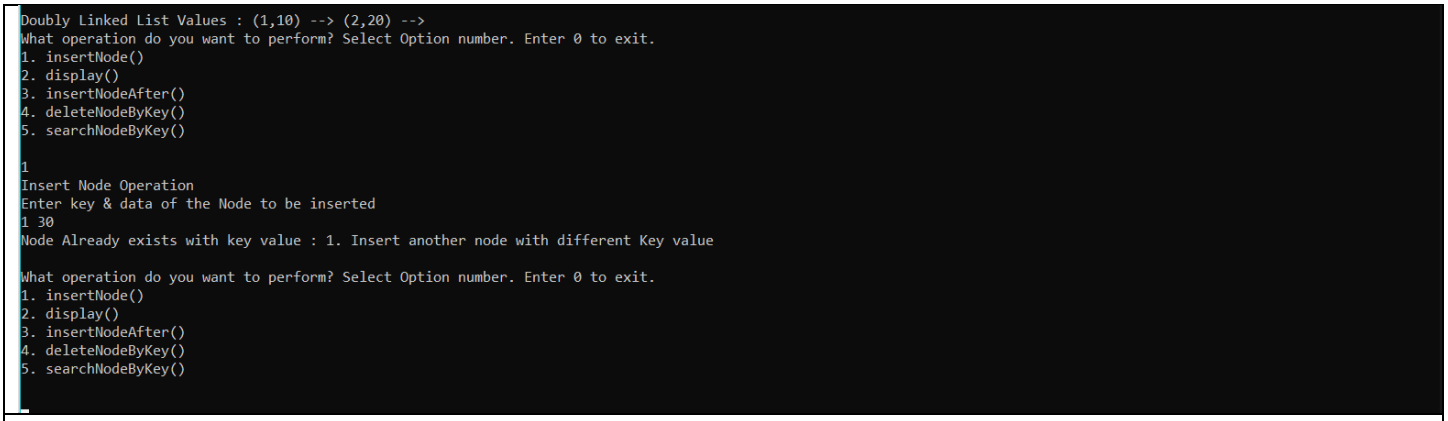
```
  }

};

int main() {

  SinglyLinkedList s;
  int option;
  int key1, k1, data1;
  do {
    cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
    cout << "1. insertNode()" << endl;
    cout << "2. display()" << endl;
    cout << "3. insertNodeAfter()" << endl;
    cout << "4. deleteNodeByKey()" << endl;
    cout << "5. searchNodeByKey()" << endl  << endl;


    cin >> option;
    Node * n1 = new Node();
    //Node n1;

    switch (option) {
    case 0:
      break;
    case 1:
      cout << "Insert Node Operation \nEnter key & data of the Node to be inserted" << endl;
      cin >> key1;
      cin >> data1;
      n1 -> key = key1;
      n1 -> data = data1;
      s.insertNode(n1);
      //cout<<n1.key<<" = "<<n1.data<<endl;
      break;


        case 2:
      s.displayList();

      break;

    case 3:
      cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New node: "
<< endl;
      cin >> k1;
      cout << "Enter key & data of the New Node first: " << endl;
      cin >> key1;
      cin >> data1;
      n1 -> key = key1;
      n1 -> data = data1;

      s.insertNodeAfter(k1, n1);
```

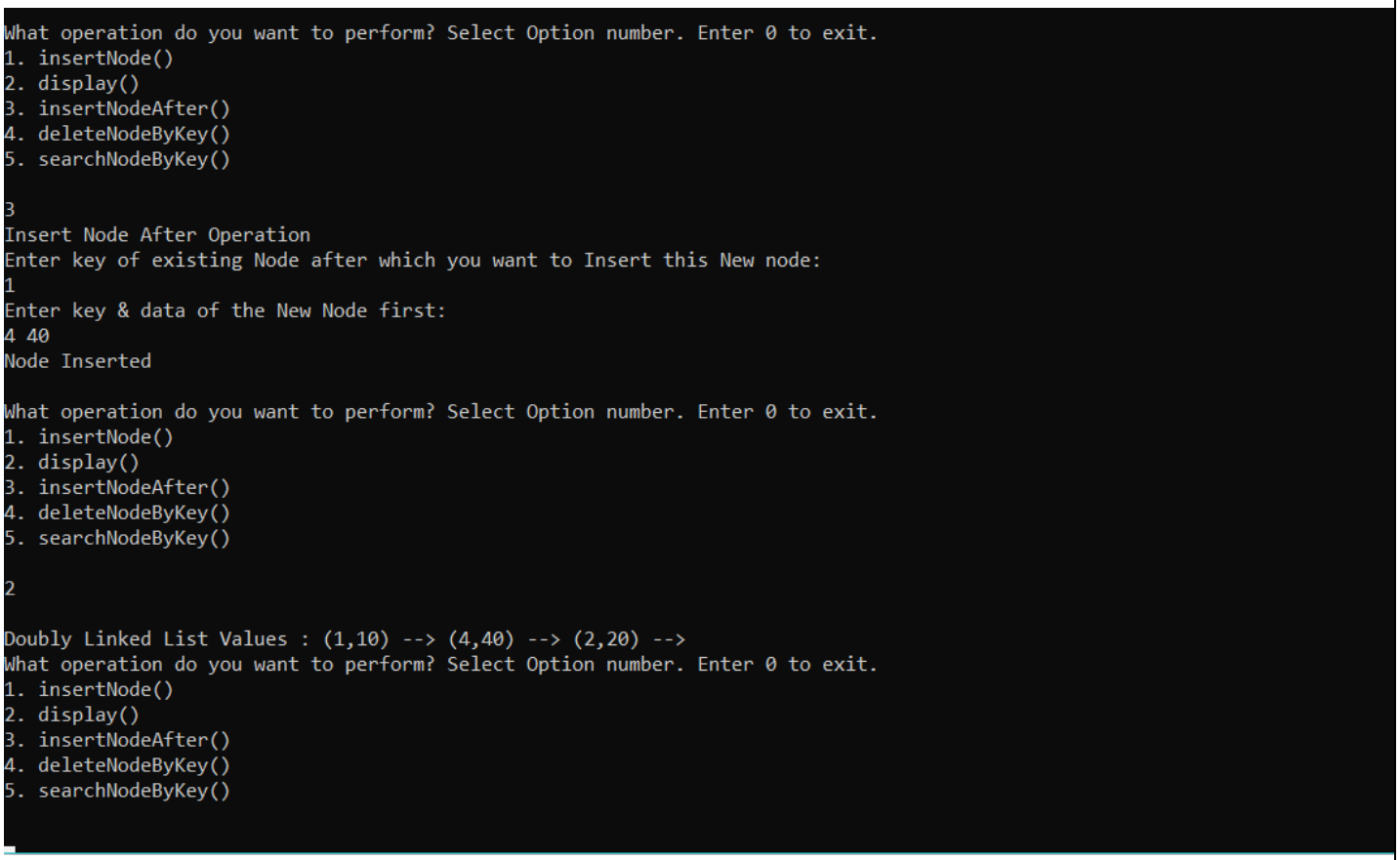
```
    break;

   case 4:

    cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
    cin >> k1;
    s.deleteNodeByKey(k1);

    break;
   case 5:
    cout << "search Node By Key Operation - \nEnter key & data to search" << endl;
    cin >> key1;
    cin >> data1;
    s.searchNodeByKey(key1, data1);

    break;


   default:
    cout << "Enter Proper Option number " << endl;
   }

 } while (option != 0);

 return 0;
}
```

Your whole Screenshot here: (Console Output):



```
"C:\Users\ASUS\Desktop\Lab task 9\number 1.exe"

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. insertNode()
2. display()
3. insertNodeAfter()
4. deleteNodeByKey()
5. searchNodeByKey()

1
Insert Node Operation
Enter key & data of the Node to be inserted
1 10
Node inserted

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. insertNode()
2. display()
3. insertNodeAfter()
4. deleteNodeByKey()
5. searchNodeByKey()

2

Singly Linked List Values : (1,10) -->
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. insertNode()
2. display()
3. insertNodeAfter()
4. deleteNodeByKey()
5. searchNodeByKey()

1
Insert Node Operation
Enter key & data of the Node to be inserted
2 20
Node inserted

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. insertNode()
2. display()
3. insertNodeAfter()
4. deleteNodeByKey()
5. searchNodeByKey()

2

Singly Linked List Values : (1,10) --> (2,20) -->
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. insertNode()
2. display()
```





2. Solve and submit at least one of the following problems. (Practice others at home)

1. Write a code to implement Doubly Linked List operations
2. Implement Stack using Linked List
3. Implement Queue using Linked List
4. Implement a program to sort the elements in a Linked List

Your code here:

Number 1

```
#include<iostream>
```

```
using namespace std;

class Node {
  public:
    int key;
  int data;
  Node * next;
  Node * previous;

  Node() {
    key = 0;
    data = 0;
    next = NULL;
    previous = NULL;
  }
  Node(int k, int d) {
    key = k;
    data = d;
  }
};

class DoublyLinkedList {

  public:
    Node * head;

  DoublyLinkedList() {
    head = NULL;
  }
  DoublyLinkedList(Node * n) {
    head = n;
  }

   // 1. CHeck if node exists using key value
  Node * nodeExists(int k) {
    Node * temp = NULL;

    Node * ptr = head;
    while (ptr != NULL) {
      if (ptr -> key == k) {
        temp = ptr;
      }
      ptr = ptr -> next;

    }
    return temp;
  }

  // 2. Insert a node to the list
  void insertNode(Node * n) {
    if (nodeExists(n -> key) != NULL) {
```

```
    cout << "Node Already exists with key value : " << n -> key << ". Insert another node with different Key value" <<
endl;
   } else {
    if (head == NULL) {
     head = n;
     cout << "Node inserted" << endl;
    } else {
     Node * ptr = head;
     while (ptr -> next != NULL) {
      ptr = ptr -> next;
     }
     ptr -> next = n;
     cout << "Node inserted" << endl;
    }
   }

  }


   // 3. Display list
  void displayList() {
   if (head == NULL) {
    cout << "No Nodes in Doubly Linked List";
   } else {
    cout << endl << "Doubly Linked List Values : ";
    Node * temp = head;

    while (temp != NULL) {
     cout << "(" << temp -> key << "," << temp -> data << ") --> ";
     temp = temp -> next;
    }
   }

  }


  // 4. Insert a Node after a given node
  void insertNodeAfter(int k, Node * n) {
   Node * ptr = nodeExists(k);
   if (ptr == NULL) {
    cout << "No node exists with key value: " << k << endl;
   } else {
    if (nodeExists(n -> key) != NULL) {
     cout << "Node Already exists with key value : " << n -> key << ". Append another node with different Key value"
<< endl;
    } else {
     n -> next = ptr -> next;
     ptr -> next = n;
     cout << "Node Inserted" << endl;
    }
   }
  }
```

```
  // 5. Delete node by unique key
  void deleteNodeByKey(int k) {
    if (head == NULL) {
      cout << "Doubly Linked List already Empty. Cant delete" << endl;
    } else if (head != NULL) {
      if (head -> key == k) {
        head = head -> next;
        cout << "Node UNLINKED with keys value : " << k << endl;
      } else {
        Node * temp = NULL;
        Node * prevptr = head;
        Node * currentptr = head -> next;
        while (currentptr != NULL) {
          if (currentptr -> key == k) {
            temp = currentptr;
            currentptr = NULL;
          } else {
            prevptr = prevptr -> next;
            currentptr = currentptr -> next;
          }
        }
        if (temp != NULL) {
          prevptr -> next = temp -> next;
          cout << "Node UNLINKED with keys value : " << k << endl;
        } else {
          cout << "Node Doesn't exist with key value : " << k << endl;
        }
      }
    }

  }
  // 6. Search node
  void searchNodeByKey(int k, int d) {

    Node * ptr = nodeExists(k);
    if (ptr != NULL) {
      ptr -> data = d;
      cout << "Node Data search Successfully" << endl;
    } else {
      cout << "Node Doesn't exist with key value : " << k << endl;
    }

  }

};

int main() {

  DoublyLinkedList s;
  int option;
  int key1, k1, data1;
  do {
    cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
```

```
    cout << "1. insertNode()" << endl;
    cout << "2. display()" << endl;
    cout << "3. insertNodeAfter()" << endl;
    cout << "4. deleteNodeByKey()" << endl;
    cout << "5. searchNodeByKey()" << endl  << endl;


    cin >> option;
    Node * n1 = new Node();
    //Node n1;

    switch (option) {
    case 0:
      break;
    case 1:
      cout << "Insert Node Operation \nEnter key & data of the Node to be inserted" << endl;
      cin >> key1;
      cin >> data1;
      n1 -> key = key1;
      n1 -> data = data1;
      s.insertNode(n1);
      //cout<<n1.key<<" = "<<n1.data<<endl;
      break;


        case 2:
      s.displayList();

      break;

    case 3:
      cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New node: "
<< endl;
      cin >> k1;
      cout << "Enter key & data of the New Node first: " << endl;
      cin >> key1;
      cin >> data1;
      n1 -> key = key1;
      n1 -> data = data1;

      s.insertNodeAfter(k1, n1);
      break;

    case 4:

      cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
      cin >> k1;
      s.deleteNodeByKey(k1);

      break;
    case 5:
      cout << "search Node By Key Operation - \nEnter key & data to search" << endl;
      cin >> key1;
```

```
    cin >> data1;
    s.searchNodeByKey(key1, data1);

    break;


  default:
    cout << "Enter Proper Option number " << endl;
  }

 } while (option != 0);

 return 0;
}
```

Your whole Screenshot here: (Console Output):