DSA BSCS 3B

Assignment 01

Submitted By

Muhammad Awais / 2883

Submitted To

Ma'am Zanaib Malik

Department of Computer Science and Engineering

National University of modern language, Islamabad

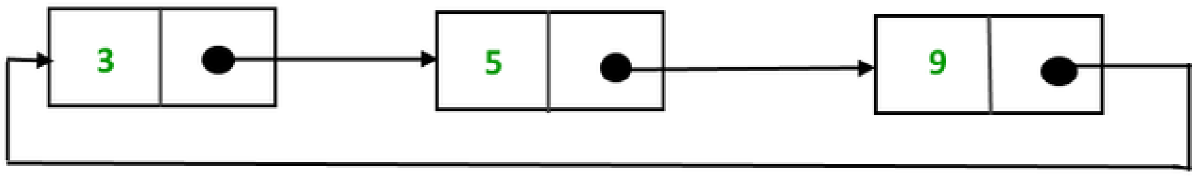
Task1 -> Discuss the properties of Circular Linked List and its application in the domain of computer science.

Properties of Circular Linked List

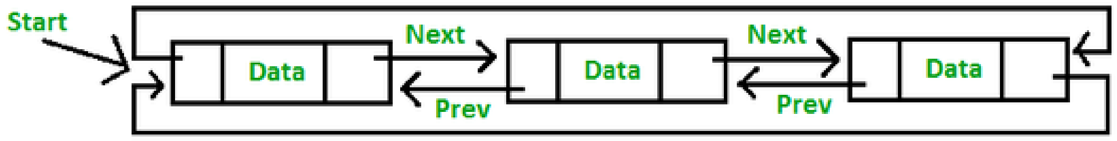
The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.

There are generally two types of circular linked lists:

1. **Circular singly linked list:** In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.



2. **Circular Doubly linked list:** Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.



Operations on the circular linked list:

We can do some operations on the circular linked list similar to the singly linked list which are:

1. Insertion
2. Deletion

1. Insertion in the circular linked list:

A node can be added in three ways:

1. Insertion at the beginning of the list
2. Insertion at the end of the list
3. Insertion in between the nodes

1) Delete the node only if it is the only node in the circular linked list:

- Free the node's memory
- The last value should be NULL A node always points to another node, so NULL assignment is not necessary.
  Any node can be set as the starting point.
  Nodes are traversed quickly from the first to the last.

2) Deletion of the last node:

- Locate the node before the last node (let it be temp)
- Keep the address of the node next to the last node in temp
- Delete the last memory
- Put temp at the end

3) Delete any node from the circular linked list: We will be given a node and our task is to delete that node from the circular linked list.

Applications of circular linked lists:

1. Multiplayer games use this to give each player a chance to play.
2. A circular linked list can be used to organize multiple running applications on an operating system. These applications are iterated over by the OS.
3. Circular linked lists can be used in resource allocation problems.
4. Circular linked lists are commonly used to implement circular buffers,
5. Circular linked lists can be used in simulation and gaming.
6. The circular linked list can be used to implement queues.

7. In web browsers, the back button is implemented using a circular linked list.
8. In an operating system, a circular linked list can be used in scheduling algorithms like the Round Robin algorithm.
9. The undo functionality that is present in applications like photo editors etc., is implemented using circular linked lists.
10. Circular linked lists can also be used to implement advanced data structures like **MRU** (Most Recently Used**) lists and Fibonacci heap.**
11. Circular lists are used in applications where the entire list is accessed one-by-one in a loop.

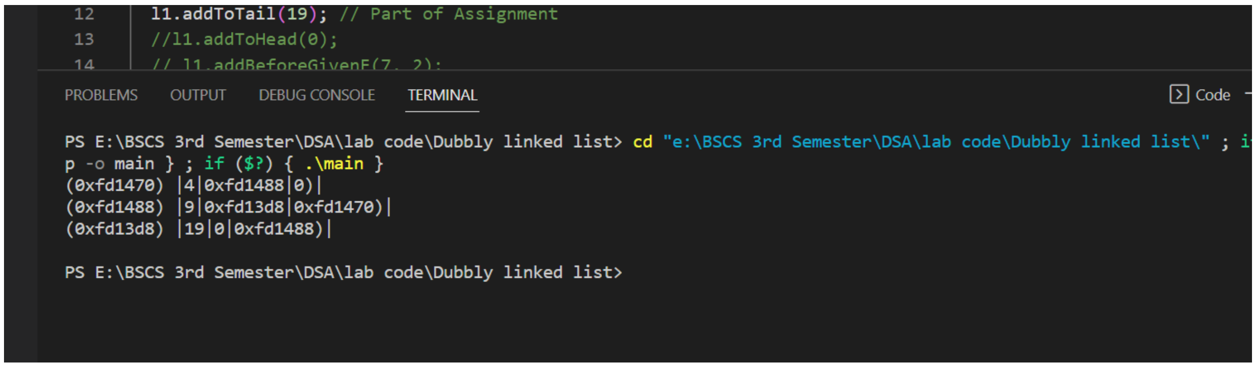
List of Task (Dubbly linked list)

1. Add to Tail
2. Add after Given Element
3. Remove from Head
4. Remove Any Given Element

1. Source Code Add to Tail

```
#include <iostream>
#include "DLL.h"
using namespace std;

int main(int argc, char **argv)
{
  DLL<int> l1;
  l1.addToTail(4);
  l1.addToTail(9);
  l1.addToTail(19);
  l1.forwardTraverse();
  cout << endl;
  return 0;
}


template <class t>
void DLL<t>::addToTail(t element)
{
    /*5 possible scenarios
1-> Error -> No
2-> Modify head only -> No
3-> Modify tail only -> if list is having one or more element
4-> Modify head and tail-> if list is empty
5-> Neither head nor tail modify -> No
 */

    DNode<t> *n = new DNode<t>(0, element, 0);
    if (head == 0 && tail == 0)
    {
        head = tail = n;
    }

    else
    {
        tail->setNext(n);
        n->setprev(tail);
        tail = n;
    }

} // End of Add To Tail
```

Output



```
l1.addToTail(19); // Part of Assignment
//l1.addToHead(0);
// l1.addBeforeGivenE(7, 2);
```

```
PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list> cd "e:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list\" ; i
p -o main } ; if ($?) { .\main }
(0xfd1470) |4|0xfd1488|0)|
(0xfd1488) |9|0xfd13d8|0xfd1470)|
(0xfd13d8) |19|0|0xfd1488)|

PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list>
```

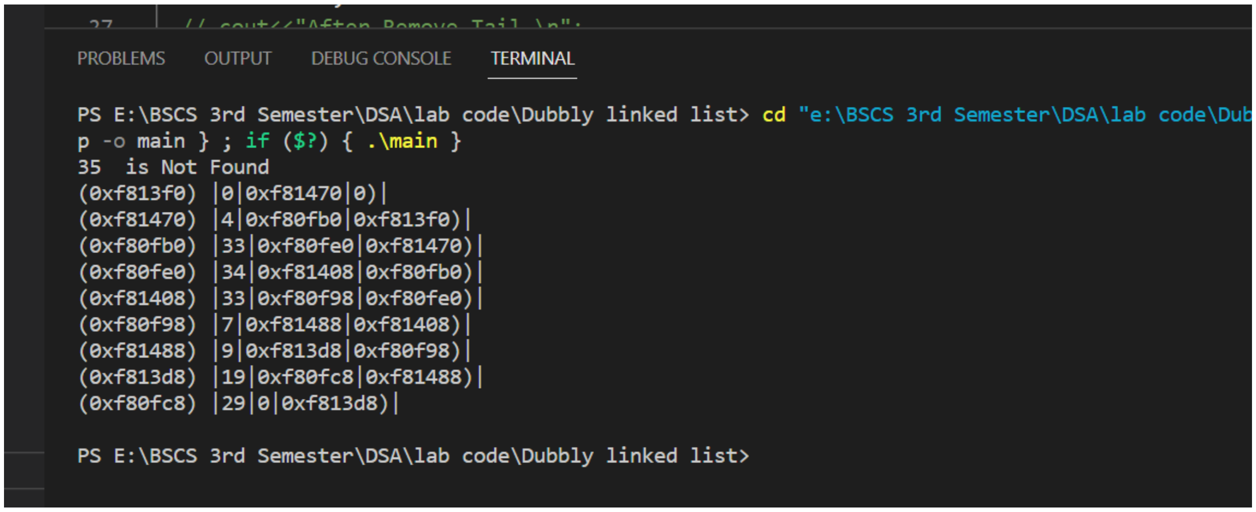
2. Source code Add After Given Element

```
#include <iostream>
#include "DLL.h"
using namespace std;

int main(int argc, char **argv)
{
  DLL<int> l1;
  // l1.addToHead(1);
  // l1.addToHead(2);
```

```
 l1.addToTail(4);
 l1.addToTail(9);
 l1.addToTail(19);
 l1.addAfterGivenE(4,33); // element on head
 l1.addAfterGivenE(19, 29);   // element on tail
 l1.addAfterGivenE(33,34); // element in between
 l1.addAfterGivenE(35,77); // element not found
 l1.forwardTraverse();

 // l1.removeFromTail();
 // l1.removeGivenElement(7); // Part of Assignment
 // l1.forwardTraverse();
 cout << endl;
 // cout<<l1.searchElement(11);  // Part of Assignment
 //  l1.reverseTraverse(); // Part Of Assignment

 return 0;
}


template <class t>
void DLL<t>::addAfterGivenE(t existingE, t newE)
{
  /*5 possible scenarios
 1-> Error -> yes if empty
 2-> Modify head only -> No
 3-> Modify tail only -> if element is on tail
 4-> Modify head and tail-> No
 5-> Neither head nor tail modify -> if element is in between of list
  */
  if (head == 0 && tail == 0)
  {
      cerr << "List is empty \n";
  }

  else if (existingE == tail->getInfo())
  {
      addToTail(newE);
  }

  else
  {
      DNode<t> *ptr = head;
      while (ptr != tail && existingE != ptr->getInfo()) // what if we use ptr!=tail
```

```cpp
        {
            ptr = (DNode<t> *)ptr->getNext();
        }

        if (ptr == tail)
        {
            cerr <<existingE<< " is Not Found \n";
        }

        else
        {
            DNode<t> *n = new DNode<t>(0, newE, 0);
            n->setNext(ptr->getNext());
            n->setprev(ptr);
            ptr->setNext(n);
            ((DNode<t> *)n->getNext())->setprev(n);
        }
    }

} // End of Add After Given Element
```

Output



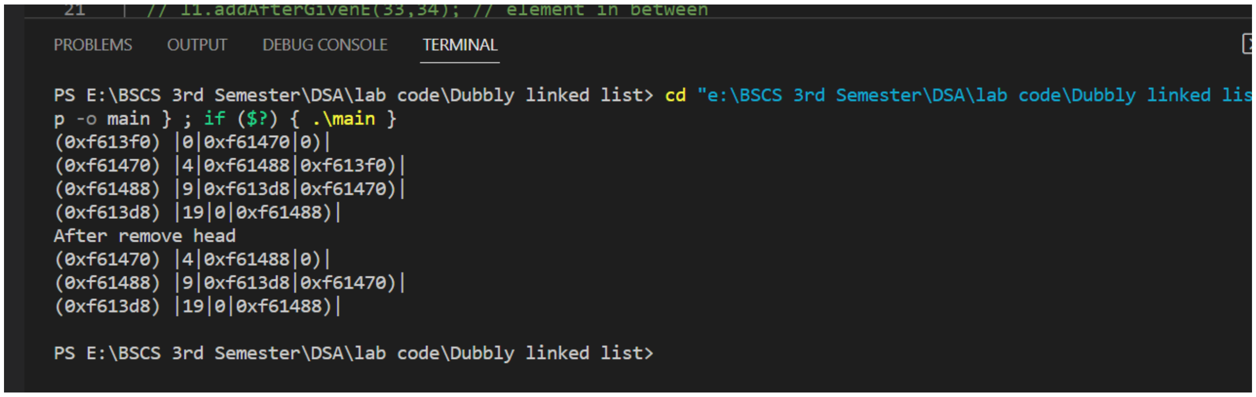
```
PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list> cd "e:\BSCS 3rd Semester\DSA\lab code\Dub
p -o main } ; if ($?) { .\main }
35  is Not Found
(0xf813f0) |0|0xf81470|0)|
(0xf81470) |4|0xf80fb0|0xf813f0)|
(0xf80fb0) |33|0xf80fe0|0xf81470)|
(0xf80fe0) |34|0xf81408|0xf80fb0)|
(0xf81408) |33|0xf80f98|0xf80fe0)|
(0xf80f98) |7|0xf81488|0xf81408)|
(0xf81488) |9|0xf813d8|0xf80f98)|
(0xf813d8) |19|0xf80fc8|0xf81488)|
(0xf80fc8) |29|0|0xf813d8)|

PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list>
```

3. Source code Remove Head

```cpp
#include <iostream>
#include "DLL.h"
using namespace std;

```

```cpp
int main(int argc, char **argv)
{
  DLL<int> l1;
  // l1.addToHead(1);
  // l1.addToHead(2);
  l1.addToTail(4);
  l1.addToTail(9);
  l1.addToTail(19);
  l1.addToHead(0);
  l1.forwardTraverse();
  cout<<"After remove head \n";
    l1.removeFromHead();
  l1.forwardTraverse();



return 0;
}


template <class t>
void DLL<t>::removeFromHead()
{
    /*5 possible scenarios
1-> Error -> yes if empty
2-> List have node (may be one or more)->Delete Head
 */
    if (head == 0 && tail == 0)
    {
        cerr << "List is empty So, nothing will delete \n";
    }
    else if (head == tail)
    {
        delete head;
        head = tail = 0;
    }
    else
    {
        DNode<t> *temp = (DNode<t> *)head->getNext();
        temp->setprev(0);
        delete head;
        head = temp;
    }

} // End of Remove from Head
```

Output

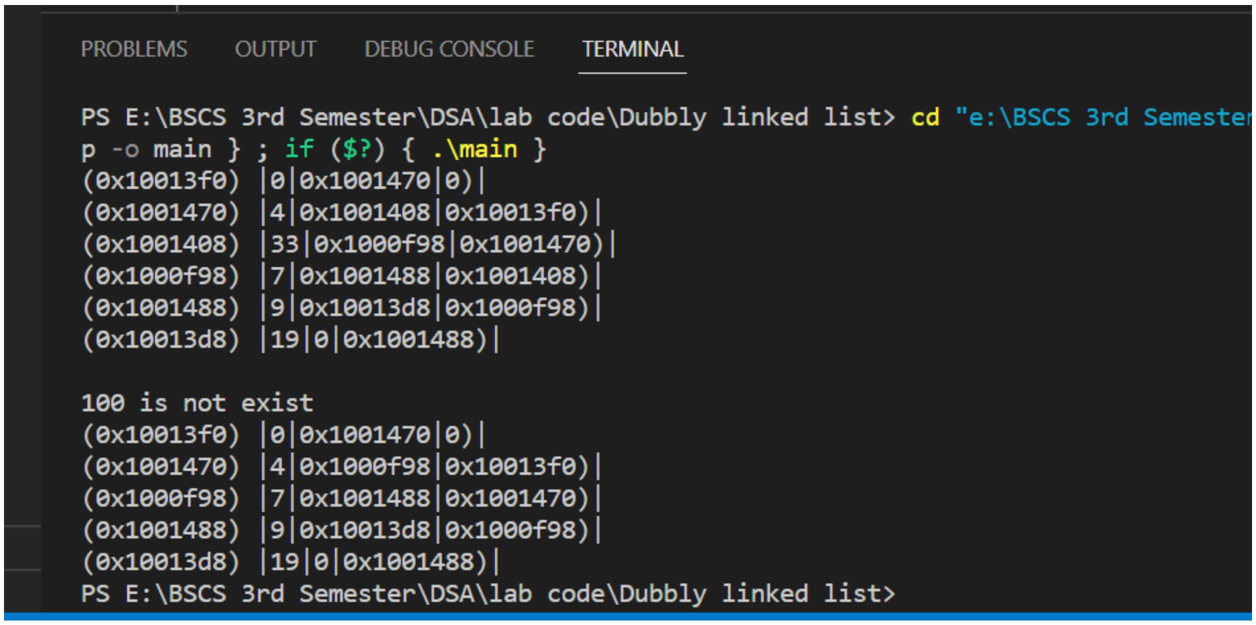


```
21  |  // l1.addAfterGivenE(33,34); // element in between
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list> cd "e:\BSCS 3rd Semester\DSA\lab code\Dubbly linked lis
p -o main } ; if ($?) { .\main }
(0xf613f0) |0|0xf61470|0)|
(0xf61470) |4|0xf61488|0xf613f0)|
(0xf61488) |9|0xf613d8|0xf61470)|
(0xf613d8) |19|0|0xf61488)|
After remove head
(0xf61470) |4|0xf61488|0)|
(0xf61488) |9|0xf613d8|0xf61470)|
(0xf613d8) |19|0|0xf61488)|

PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list>
```

4. Source code Remove Given Element

```
#include <iostream>
#include "DLL.h"
using namespace std;

int main(int argc, char **argv)
{
  DLL<int> l1;
  // l1.addToHead(1);
  // l1.addToHead(2);
  l1.addToTail(4);
  l1.addToTail(9);
  l1.addToTail(19);
  l1.addToHead(0);
  l1.addAfterGivenE(4,33);
  l1.addBeforeGivenE(7, 9);
 l1.forwardTraverse();
  // l1.removeFromTail();
   l1.removeGivenElement(33); // Part of Assignment
   l1.removeGivenElement(100);
    l1.forwardTraverse();
  // l1.forwardTraverse();
  cout << endl;
  // cout<<l1.searchElement(11);  // Part of Assignment
  //  l1.reverseTraverse(); // Part Of Assignment

  return 0;
}


template <class t>
void DLL<t>::removeGivenElement(t element)
{
    /*5 possible scenarios
1-> Error -> yes if empty
2-> Delete head only -> if element is on head
3-> Delete tail only -> if element is on tail
4-> Delete head and tail-> only one element in list
5-> Neither head nor tail delete -> if element is in between of list
 */

    if (head == 0 && tail == 0) // empty
    {
        cerr << "List is empty, Deletion can not possible \n";
    }

    else if (head == tail) // delete head and tail
    {
        delete head;
        head = tail = 0;
    }

    else if (element == head->getInfo()) // delete head
    {
        removeFromHead();
    }

    else if (element == tail->getInfo()) // delete tail
    {
        removeFromTail();
    }

    else // // delete neither head nor tail
    {
        DNode<t> *ptr = head;

        while (ptr != tail && element != ptr->getNext()->getInfo()) // what if
element on tail
        {
            ptr = (DNode<t> *)ptr->getNext();
        }

        if (ptr == tail)
        {
            cerr <<element<< " is not exist \n";
        }

        else
        {
            // ptr->setNext(ptr->getNext()->getnext())
            DNode<t> *temp = ((DNode<t> *)ptr->getNext()->getNext());
            temp->setprev(ptr);
            delete ptr->getNext();
            ptr->setNext(temp);
        }
    }

} // End of Remove Given Element
```

Output



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list> cd "e:\BSCS 3rd Semester
p -o main } ; if ($?) { .\main }
(0x10013f0) |0|0x1001470|0)|
(0x1001470) |4|0x1001408|0x10013f0)|
(0x1001408) |33|0x1000f98|0x1001470)|
(0x1000f98) |7|0x1001488|0x1001408)|
(0x1001488) |9|0x10013d8|0x1000f98)|
(0x10013d8) |19|0|0x1001488)|

100 is not exist
(0x10013f0) |0|0x1001470|0)|
(0x1001470) |4|0x1000f98|0x10013f0)|
(0x1000f98) |7|0x1001488|0x1001470)|
(0x1001488) |9|0x10013d8|0x1000f98)|
(0x10013d8) |19|0|0x1001488)|
PS E:\BSCS 3rd Semester\DSA\lab code\Dubbly linked list>
```

End