UTM

UNIVERSITI TEKNOLOGI MALAYSIA

SECJ2013 DATA STRUCTURE AND ALGORITHM
SECTION 6
LAB EXERCISE 3

Name	Brendan Dylan Gampa Anak Joseph Dusit @ Dusit (A20EC0021)
Matric No	A20EC0021

LECTURER: TS. DR. JOHANNA BINTI AHMAD

SUBMISSION DATE: 4th JANUARY 2022

LAB EXERCISE 3

QUESTION 1 (CIRCULAR DOUBLY LINKED LIST)

```
/*
 * C++ Program to Implement Circular Doubly Linked List
 */
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;

/*
 * Node Declaration
 */
struct node
{
    int info;
    struct node *next;
    struct node *prev;
}*start, *last;
int counter = 0;
/*
 * Class Declaration
 */
class double_clist
{
    public:
        node *create_node(int);
        void insert_begin();
        void insert_last();
        void insert_pos();
        void delete_pos();
        void search();
        void update();
        void display();
        void reverse();
        void sort();
        double_clist()
        {
            start = NULL;
            last = NULL;
        }
};

/*
```

```
 * Main: Contains Menu
 */
int main()
{
    int choice;
    double_clist cdl;
    while (1)
    {
        cout<<"\n--------------------------------"<<endl;
        cout<<"Operations on Doubly Circular linked list"<<endl;
        cout<<"\n--------------------------------"<<endl;
        cout<<"1.Insert at Beginning"<<endl;
        cout<<"2.Insert at Last"<<endl;
        cout<<"3.Insert at Position"<<endl;
        cout<<"4.Delete at Position"<<endl;
        cout<<"5.Update Node"<<endl;
        cout<<"6.Search Element"<<endl;
        cout<<"7.Sort"<<endl;
        cout<<"8.Display List"<<endl;
        cout<<"9.Reverse List"<<endl;
        cout<<"10.Exit"<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cdl.insert_begin();
            break;
        case 2:
            cdl.insert_last();
            break;
        case 3:
            cdl.insert_pos();
            break;
        case 4:
            cdl.delete_pos();
            break;
        case 5:
            cdl.update();
            break;
        case 6:
            cdl.search();
            break;
        case 7:
            cdl.sort();
            break;
```

```
        case 8:
            cdl.display();
            break;
        case 9:
            cdl.reverse();
            break;
        case 10:
            exit(1);
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
    return 0;
}

/*
 *MEMORY ALLOCATED FOR NODE DYNAMICALLY
 */
node* double_clist::create_node(int value)
{
    counter++;
    struct node *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
/*
 *INSERTS ELEMENT AT BEGINNING
 */
void double_clist::insert_begin()
{
    int value;
    cout<<endl<<"Enter the element to be inserted: ";
    cin>>value;
    struct node *temp;
    temp = create_node(value);
    if (start == last && start == NULL)
    {
        cout<<"Element inserted in empty list"<<endl;
        start = last = temp;
        start->next = last->next = NULL;
        start->prev = last->prev = NULL;
    }
    else
```

```
    {
        temp->next = start;
        start->prev = temp;
        start = temp;
        start->prev = last;
        last->next = start;
        cout<<"Element inserted"<<endl;
    }
}

/*
 *INSERTS ELEMNET AT LAST
 */
void double_clist::insert_last()
{
    int value;
    cout<<endl<<"Enter the element to be inserted: ";
    cin>>value;
    struct node *temp;
    temp = create_node(value);
    if (start == last && start == NULL)
    {
        cout<<"Element inserted in empty list"<<endl;
        start = last = temp;
        start->next = last->next = NULL;
        start->prev = last->prev = NULL;
    }
    else
    {
        last->next = temp;
        temp->prev = last;
        last = temp;
        start->prev = last;
        last->next = start;
    }
}
/*
 *INSERTS ELEMENT AT POSITION
 */
void double_clist::insert_pos()
{
    int value, pos, i;
    cout<<endl<<"Enter the element to be inserted: ";
    cin>>value;
    cout<<endl<<"Enter the postion of element inserted: ";
    cin>>pos;
```

```
    struct node *temp, *s, *ptr;
    temp = create_node(value);
    if (start == last && start == NULL)
    {
        if (pos == 1)
        {
            start = last = temp;
            start->next = last->next = NULL;
            start->prev = last->prev = NULL;
        }
        else
        {
            cout<<"Position out of range"<<endl;
            counter--;
            return;
        }
    }
    else
    {
        if (counter < pos)
        {
             cout<<"Position out of range"<<endl;
             counter--;
             return;
        }
        s = start;
        for (i = 1;i <= counter;i++)
        {
            ptr = s;
            s = s->next;
            if (i == pos - 1)
            {
                ptr->next = temp;
                temp->prev = ptr;
                temp->next = s;
                s->prev = temp;
                cout<<"Element inserted"<<endl;
                break;
            }
        }
    }
}
/*
 * Delete Node at Particular Position
 */
void double_clist::delete_pos()
```

```
{
    int pos, i;
    node *ptr, *s;
    if (start == last && start == NULL)
    {
        cout<<"List is empty, nothing to delete"<<endl;
        return;
    }
    cout<<endl<<"Enter the postion of element to be deleted: ";
    cin>>pos;
    if (counter < pos)
    {
        cout<<"Position out of range"<<endl;
        return;
    }
    s = start;
    if(pos == 1)
    {
        counter--;
        last->next = s->next;
        s->next->prev = last;
        start = s->next;
        free(s);
        cout<<"Element Deleted"<<endl;
        return;
    }
    for (i = 0;i < pos - 1;i++ )
    {
        s = s->next;
        ptr = s->prev;
    }
    ptr->next = s->next;
    s->next->prev = ptr;
    if (pos == counter)
    {
        last = ptr;
    }
    counter--;
    free(s);
    cout<<"Element Deleted"<<endl;
}
/*
 * Update value of a particular node
 */
void double_clist::update()
{
```

```
    int value, i, pos;
    if (start == last && start == NULL)
    {
        cout<<"The List is empty, nothing to update"<<endl;
        return;
    }
    cout<<endl<<"Enter the postion of node to be updated: ";
    cin>>pos;
    cout<<"Enter the new value: ";
    cin>>value;
    struct node *s;
    if (counter < pos)
    {
        cout<<"Position out of range"<<endl;
        return;
    }
    s = start;
    if (pos == 1)
    {
       s->info = value;
       cout<<"Node Updated"<<endl;
       return;
    }
    for (i=0;i < pos - 1;i++)
    {
        s = s->next;
    }
    s->info = value;
    cout<<"Node Updated"<<endl;
}
/*
 * Search Element in the list
 */
void double_clist::search()
{
    int pos = 0, value, i;
    bool flag = false;
    struct node *s;
    if (start == last && start == NULL)
    {
        cout<<"The List is empty, nothing to search"<<endl;
        return;
    }
    cout<<endl<<"Enter the value to be searched: ";
    cin>>value;
    s = start;
```

```
    for (i = 0;i < counter;i++)
    {
        pos++;
        if (s->info == value)
        {
            cout<<"Element "<<value<<" found at position: "<<pos<<endl;
            flag = true;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element not found in the list"<<endl;
}
/*
 * Sorting Doubly Circular Link List
 */
void double_clist::sort()
{
    struct node *temp, *s;
    int value, i;
    if (start == last && start == NULL)
    {
        cout<<"The List is empty, nothing to sort"<<endl;
        return;
    }
    s = start;
    for (i = 0;i < counter;i++)
    {
        temp = s->next;
        while (temp != start)
        {
            if (s->info > temp->info)
            {
                value = s->info;
                s->info = temp->info;
                temp->info = value;
            }
            temp = temp->next;
        }
        s = s->next;
    }
}
/*
 * Display Elements of the List
 */
void double_clist::display()
```

```
{
    int i;
    struct node *s;
    if (start == last && start == NULL)
    {
        cout<<"The List is empty, nothing to display"<<endl;
        return;
    }
    s = start;
    for (i = 0;i < counter-1;i++)
    {
        cout<<s->info<<"<->";
        s = s->next;
    }
    cout<<s->info<<endl;
}
/*
 * Reverse Doubly Circular Linked List
 */
void double_clist::reverse()
{
    if (start == last && start == NULL)
    {
        cout<<"The List is empty, nothing to reverse"<<endl;
        return;
    }
    struct node *p1, *p2;
    p1 = start;
    p2 = p1->next;
    p1->next = NULL;
    p1->prev = p2;
    while (p2 != start)
    {
        p2->prev = p2->next;
        p2->next = p1;
        p1 = p2;
        p2 = p2->prev;
    }
    last = start;
    start = p1;
    cout<<"List Reversed"<<endl;
}
```

QUESTION

1. Insert 5 values then attach the output for **operation 1 to 9.**

Operation 1 (Insert at beginning)

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 1

Enter the element to be inserted: 1
Element inserted in empty list
```

Operation 2 (Insert at Last)

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 2

Enter the element to be inserted: 5
```

Operation 3 (Insert 3 elements into the nodes)

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 3

Enter the element to be inserted: 2

Enter the postion of element inserted: 2
Element inserted
```

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 3

Enter the element to be inserted: 4

Enter the postion of element inserted: 4
Element inserted
```

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 3

Enter the element to be inserted: 6

Enter the postion of element inserted: 6
Element inserted
```

Operation 4 (Delete Node)

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 4

Enter the postion of element to be deleted: 6
Element Deleted
```

Operation 5 (Update Node)

```
--------------------------------
Operations on Doubly Circular linked list

--------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 5

Enter the postion of node to be updated: 4
Enter the new value: 7
Node Updated
```

Operation 6 (Search Element)

***if found**

```
------------------------------
Operations on Doubly Circular linked list

------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 6

Enter the value to be searched: 3
Element 3 found at position: 3
```

***if not found**

```
------------------------------
Operations on Doubly Circular linked list

------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 6

Enter the value to be searched: 8
Element not found in the list
```

Operation 7 (Sort)

```
------------------------------
Operations on Doubly Circular linked list

------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 7
```

Operation 8 (Display list)

```
------------------------------
Operations on Doubly Circular linked list

------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 8
1<->2<->3<->5<->7
```

Operation 9 (Reverse Sort)

```
------------------------------
Operations on Doubly Circular linked list

------------------------------
1.Insert at Beginning
2.Insert at Last
3.Insert at Position
4.Delete at Position
5.Update Node
6.Search Element
7.Sort
8.Display List
9.Reverse List
10.Exit
Enter your choice : 9
List Reversed
```

***output**

```
Enter your choice : 8
7<->5<->3<->2<->1
```

Question 2 (Circular Linked List)

```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
   int data;                                                    //data of
the node
   struct Node *next;                                           //pointer
to the next node in the list
};

int Length(struct Node *head)                                   //function to
calculate the length of the linked list
{
   struct Node *t;
   int i = 0;
   if (head == NULL)
            //handle underflow condition
   {
      return 0;
   }

   t = head -> next;

   do

   {
                                    //handle traversal through the list
      t = t -> next;
      i++;
   } while (t != head->next);
   return i;
}

struct Node *Start(struct Node *head, int data)                 //function to insert
nodes at the beginning of the list
{
   struct Node *temp = (struct Node *) malloc(sizeof(struct Node));
   if (head == NULL)
                                    //handle underflow condition
      {
   temp -> data = data;
   head = temp;
   head -> next = head;
      }
   else
   {
```

```
    temp -> data = data;
    temp -> next = head -> next;
    head -> next = temp;
    head = temp;
    }
    return head;
}

struct Node *End(struct Node *head, int data)
            //function to insert nodes at the end of the list
{
    struct Node *temp = (struct Node *) malloc(sizeof(struct Node)), *a = head;
    if (head == NULL)
                                    //handle underflow condition
        {
    temp -> data = data;
    head = temp;
    head -> next = head;
        }
    else
    {
    do
    {
        a = a -> next;
    } while (a -> next != head);
            //traverse to the end of the list
    temp -> data = data;
    temp -> next = a -> next;
    a -> next = temp;
    }
    return head;
}

struct Node *Middle(struct Node *head, int data, int index)
            //function to insert nodes anywhere in between the beginning and the end
{
    if (head == NULL)
            //handle underflow condition
    {
        cout << "List is empty!" << endl;
        return NULL;
    }

    int len = Length(head);                                         //get the length of
the list for making a decision
    if (index > len || index < 0)                                   //wrong index
given
    {
        cout << "Wrong input, insertion not possible!" << endl;
```

```
        return head;
    }
    if (index == 0)                                                    //insert
at the beginning
    {
        head = Start(head,data);
        return head;
    }
    if (index == len)                                                  //insert
at the end
    {
        head = End(head,data);
        return head;
    }
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node)), *a = head;
    do

    {
                                          //handle data traversal from beginning to end
        a = a -> next;
    } while (a -> next != head);
    len = 0;
    while (1)
    {
        if (len == index)                                              //handle
node addition
        {
            temp -> data = data;
            temp -> next = a -> next;
            a -> next = temp;
            return head;
        }
        a = a -> next;
        len++;
    }
}

struct Node *First(struct Node* head)
                                  //function to delete node from the start of the list
{
    struct Node *prev = head, *first = head;

    if (head == NULL) {
              //handle underflow condition
        cout << "List is empty" << endl;
        return NULL;
    }
```

```
    if (prev->next == prev)
                        //handle deletion for only one node of the list
    {
        head = NULL;
        return head;
    }

    while (prev->next != head)
    {
        prev = prev->next;
    }

    prev->next = first->next;

    head = prev->next;
    free(first);
    return head;
}

struct Node *Last(struct Node* head)
                        //function to delete node from the end of the list
{
    struct Node *curr = head, *temp = head, *prev;

    if (head == NULL) {
                                //handle underflow condition
        cout << "List is empty" << endl;
        return NULL;
    }

    if (curr->next == curr)
                        //handle deletion for only one node of the list
    {
        head = NULL;
        return head;
    }

    while (curr->next != head)
    {
        prev = curr;
        curr = curr->next;
    }

    prev->next = curr->next;
    head = prev->next;
    free(curr);
    return head;
}
```

```
struct Node *Position(struct Node* head, int index)
            //function to delete data from anywhere in between the start and the end of the list
{
   int len = Length(head);
   int count = 1;
   struct Node *prev = head, *next = head;

   if (head == NULL) {
                                    //handle underflow condition
      cout << "List is empty" << endl;
      return NULL;
   }

   if (index > len || index < 0)
            //wrong index entered
   {
      cout << "Wrong data given, deletion not possible!" << endl;
      return head;
   }

   if (index == 0)
   {
      First(head);
      return head;
   }

   if (index == len)
   {
      Last(head);
      return head;
   }

   while (len > 0)
   {
                                          //traverse through the list and delete the node in the list
      if (index == count)
      {
         prev->next = next->next;
         free(next);
         return head;
      }
      prev = prev->next;
      next = prev->next;
      len--;
      count++;
   }

   return head;
}
```

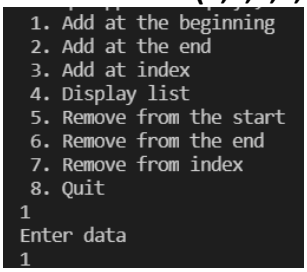
```
void Display(struct Node *head)
                                        //function to traverse throughout the list
{
   struct Node *t;
   if (head == NULL)
   {
      cout << "Linked list is empty." << endl;
      return;
   }

   t = head;

   do
   {
      cout << t -> data << " -> ";
      t = t -> next;
   } while (t != head);
   cout<<" "<<endl;
}

int main()
{
   struct Node *head = NULL;
   int n = 0,a = 0;
   char ch;
   while (1)
   {
   cout << " 1. Add at the beginning\n 2. Add at the end \n 3. Add at index \n 4. Display list \n 5.
Remove from the start \n 6. Remove from the end \n 7. Remove from index \n 5. Quit" << endl;
   cin >> ch;
   switch (ch)
   {
      case '1' :  cout << "Enter data" << endl;
                  cin >> n;
                  head = Start(head, n);
                  break;
      case '2' :  cout << "Enter data" << endl;
                  cin >> n;
                  head = End(head, n);
                  break;
      case '3' :  cout << "Enter data" << endl;
                  cin >> n;
                  cout << "Enter index to insert at" << endl;
                  cin >> a;
                  head = Middle(head, n, a);
                  break;
      case '4' :  Display(head);
                  break;
```

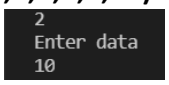
```
        case '5' :  head = First(head);
                    break;
        case '6' :  head = Last(head);
                    break;
        case '7' :  cout << "Enter index to insert at" << endl;
                    cin >> a;
                    head = Position(head,a);
                    break;
        case '8' : exit(0);
        default : cout << "Wrong choice!" << endl;
    }
    }
    return 0;
}
```

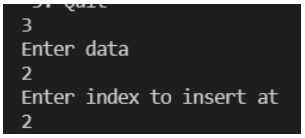
QUESTION

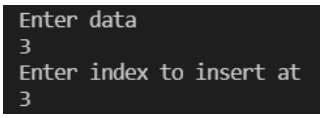
1. **Insert 10 values** then attach the output.

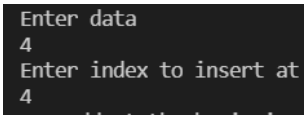
Insert 10 data (1,2,3,4,5,6,7,8,9,10)

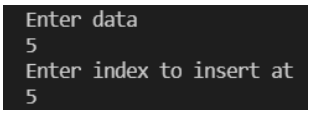


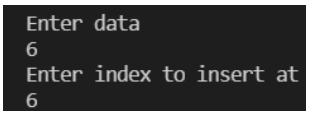


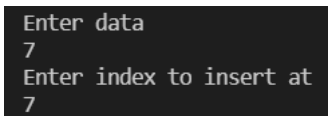


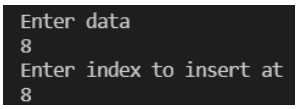


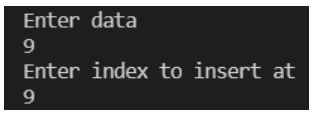




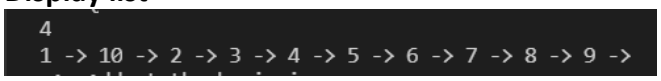




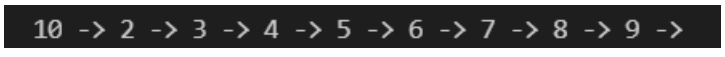




Display list



Remove from start



Remove from end

Remove from index