Name : Bhushan Sharad Tejankar

Roll no. : I30

reg_no. : 2020BIT030

```cpp
// DAA Practical 1
// 1. Stack
#include<iostream>
#include<string>
using namespace std;
class Stack {
  private:
    int top;
  int arr[5];

  public:
    Stack() {
      top = -1;
      for (int i = 0; i < 5; i++) {
        arr[i] = 0;
      }
    }
  bool isEmpty() {
    if (top == -1)
      return true;
    else
      return false;
  }

  bool isFull() {
    if (top == 4)
      return true;
    else
      return false;
  }
```

```
31
32  void push(int val) {
33    if (isFull()) {
34      cout << "stack overflow" << endl;
35    } else {
36      top++; // 1
37      arr[top] = val;
38    }
39  }
40
41  int pop() {
42    if (isEmpty()) {
43      cout << "stack underflow" << endl;
44      return 0;
45    } else {
46      int popValue = arr[top];
47      arr[top] = 0;
48      top--;
49      return popValue;
50    }
51  }
52
53  int count() {
54    return (top + 1);
55  }
56
57  int peek(int pos) {
58    if (isEmpty()) {
59      cout << "stack underflow" << endl;
60      return 0;
```

```
61      } else {
62        return arr[pos];
63      }
64    }
65
66    void change(int pos, int val) {
67      arr[pos] = val;
68      cout << "value changed at location " << pos << endl;
69    }
70
71    void display() {
72      cout << "All values in the Stack are " << endl;
73      for (int i = 4; i >= 0; i--) {
74        cout << arr[i] << endl;
75      }
76    }
77  };
78
79  int main() {
80    Stack s1;
81    int option, postion, value;
82
83    do {
84      cout << "What operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
85      cout << "1. Push()" << endl;
86      cout << "2. Pop()" << endl;
87      cout << "3. isEmpty()" << endl;
88      cout << "4. isFull()" << endl;
89      cout << "5. peek()" << endl;
90      cout << "6. count()" << endl;
```

```
91      cout << "7. change()" << endl;
92      cout << "8. display()" << endl;
93      cout << "9. Clear Screen" << endl << endl;
94
95      cin >> option;
96      switch (option) {
97      case 0:
98        break;
99      case 1:
100       cout << "Enter an item to push in the stack" << endl;
101       cin >> value;
102       s1.push(value);
103       break;
104     case 2:
105       cout << "Pop Function Called - Poped Value: " << s1.pop() << endl;
106       break;
107     case 3:
108       if (s1.isEmpty())
109         cout << "Stack is Empty" << endl;
110       else
111         cout << "Stack is not Empty" << endl;
112       break;
113     case 4:
114       if (s1.isFull())
115         cout << "Stack is Full" << endl;
116       else
117         cout << "Stack is not Full" << endl;
118       break;
119     case 5:
120       cout << "Enter position of item you want to peek: " << endl;
```

```
121         cin >> postion;
122         cout << "Peek Function Called - Value at position " << postion << " is " << s1.peek(postion) << endl;
123         break;
124       case 6:
125         cout << "Count Function Called - Number of Items in the Stack are: " << s1.count() << endl;
126         break;
127       case 7:
128         cout << "Change Function Called - " << endl;
129         cout << "Enter position of item you want to change : ";
130         cin >> postion;
131         cout << endl;
132         cout << "Enter value of item you want to change : ";
133         cin >> value;
134         s1.change(postion, value);
135         break;
136       case 8:
137         cout << "Display Function Called - " << endl;
138         s1.display();
139         break;
140       case 9:
141         system("cls");
142         break;
143       default:
144         cout << "Enter Proper Option number " << endl;
145       }
146
147     } while (option != 0);
148
```

```cpp
// 2. Queue
#include<iostream>
using namespace std;
class Queue {
  private:
    int front;
  int rear;
  int arr[5];

  public:
    Queue() {
      front = -1;
      rear = -1;
      for (int i = 0; i < 5; i++) {
        arr[i] = 0;
      }
    }
  bool isEmpty() {
    if (front == -1 && rear == -1)
      return true;
    else
      return false;
  }
  bool isFull() {
    if (rear == 4)
      return true;
    else
      return false;
  }
  void enqueue(int val) {
```

```
    if (isFull()) {
      cout << "Queue full" << endl;
      return;
    } else if (isEmpty()) {
      rear = 0;
      front = 0;
      arr[rear] = val;
    } else {
      rear++;
      arr[rear] = val;
    }

  }

  int dequeue() {
    int x = 0;
    if (isEmpty()) {
      cout << "Queue is Empty" << endl;
      return x;
    } else if (rear == front) {
      x = arr[rear];
      rear = -1;
      front = -1;
      return x;
    } else {
      cout << "front value: " << front << endl;
      x = arr[front];
      arr[front] = 0;
      front++;
      return x;
```

```
61      }
62    }
63
64    int count() {
65      return (rear - front + 1);
66    }
67
68    void display() {
69      cout << "All values in the Queue are - " << endl;
70      for (int i = 0; i < 5; i++) {
71        cout << arr[i] << "  ";
72      }
73    }
74
75  };
76
77  int main() {
78    Queue q1;
79    int value, option;
80
81    do {
82      cout << "\n\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
83      cout << "1. Enqueue()" << endl;
84      cout << "2. Dequeue()" << endl;
85      cout << "3. isEmpty()" << endl;
86      cout << "4. isFull()" << endl;
87      cout << "5. count()" << endl;
88      cout << "6. display()" << endl;
89      cout << "7. Clear Screen" << endl << endl;
90
```

```
91      cin >> option;
92
93      switch (option) {
94      case 0:
95        break;
96      case 1:
97        cout << "Enqueue Operation \nEnter an item to Enqueue in the Queue" << endl;
98        cin >> value;
99        q1.enqueue(value);
100       break;
101     case 2:
102       cout << "Dequeue Operation \nDequeued Value : " << q1.dequeue() << endl;
103       break;
104     case 3:
105       if (q1.isEmpty())
106         cout << "Queue is Empty" << endl;
107       else
108         cout << "Queue is not Empty" << endl;
109       break;
110     case 4:
111       if (q1.isFull())
112         cout << "Queue is Full" << endl;
113       else
114         cout << "Queue is not Full" << endl;
115       break;
116     case 5:
117       cout << "Count Operation \nCount of items in Queue : " << q1.count() << endl;
118       break;
119     case 6:
120       cout << "Display Function Called - " << endl;
```

```
121         q1.display();
122         break;
123       case 7:
124         system("cls");
125         break;
126       default:
127         cout << "Enter Proper Option number " << endl;
128       }
129
130   } while (option != 0);
131
132   return 0;
133
134 }
135
136
137
```

```cpp
// 3. Linked List

#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;
```

```
36
37      Node * ptr = head;
38      while (ptr != NULL) {
39        if (ptr -> key == k) {
40          temp = ptr;
41        }
42        ptr = ptr -> next;
43
44      }
45      return temp;
46    }
47
48    // 2. Append a node to the list
49    void appendNode(Node * n) {
50      if (nodeExists(n -> key) != NULL) {
51        cout << "Node Already exists with key value : " << n -> key << ". Append another node with different Key value" << endl;
52      } else {
53        if (head == NULL) {
54          head = n;
55          cout << "Node Appended" << endl;
56        } else {
57          Node * ptr = head;
58          while (ptr -> next != NULL) {
59            ptr = ptr -> next;
60          }
61          ptr -> next = n;
62          cout << "Node Appended" << endl;
63        }
64      }
65
66    }
67    // 3. Prepend Node - Attach a node at the start
68    void prependNode(Node * n) {
69      if (nodeExists(n -> key) != NULL) {
70        cout << "Node Already exists with key value : " << n -> key << ". Append another node with different Key value" << endl;
```

```
71      } else {
72        n -> next = head;
73        head = n;
74        cout << "Node Prepended" << endl;
75      }
76    }
77
78    // 4. Insert a Node after a particular node in the list
79    void insertNodeAfter(int k, Node * n) {
80      Node * ptr = nodeExists(k);
81      if (ptr == NULL) {
82        cout << "No node exists with key value: " << k << endl;
83      } else {
84        if (nodeExists(n -> key) != NULL) {
85          cout << "Node Already exists with key value : " << n -> key << ". Append another node with different Key value" << endl;
86        } else {
87          n -> next = ptr -> next;
88          ptr -> next = n;
89          cout << "Node Inserted" << endl;
90        }
91      }
92    }
93
94    // 5. Delete node by unique key
95    void deleteNodeByKey(int k) {
96      if (head == NULL) {
97        cout << "Singly Linked List already Empty. Cant delete" << endl;
98      } else if (head != NULL) {
99        if (head -> key == k) {
100         head = head -> next;
101         cout << "Node UNLINKED with keys value : " << k << endl;
102       } else {
103         Node * temp = NULL;
104         Node * prevptr = head;
105         Node * currentptr = head -> next;
```

```
106         while (currentptr != NULL) {
107           if (currentptr -> key == k) {
108             temp = currentptr;
109             currentptr = NULL;
110           } else {
111             prevptr = prevptr -> next;
112             currentptr = currentptr -> next;
113           }
114         }
115         if (temp != NULL) {
116           prevptr -> next = temp -> next;
117           cout << "Node UNLINKED with keys value : " << k << endl;
118         } else {
119           cout << "Node Doesn't exist with key value : " << k << endl;
120         }
121       }
122     }
123
124   }
125   // 6th update node
126   void updateNodeByKey(int k, int d) {
127
128     Node * ptr = nodeExists(k);
129     if (ptr != NULL) {
130       ptr -> data = d;
131       cout << "Node Data Updated Successfully" << endl;
132     } else {
133       cout << "Node Doesn't exist with key value : " << k << endl;
134     }
135
136   }
137
138   // 7th printing
139   void printList() {
140     if (head == NULL) {
```

```
141         cout << "No Nodes in Singly Linked List";
142       } else {
143         cout << endl << "Singly Linked List Values : ";
144         Node * temp = head;
145 
146         while (temp != NULL) {
147           cout << "(" << temp -> key << "," << temp -> data << ") --> ";
148           temp = temp -> next;
149         }
150       }
151 
152     }
153 
154 };
155 
156 int main() {
157 
158   SinglyLinkedList s;
159   int option;
160   int key1, k1, data1;
161   do {
162     cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
163     cout << "1. appendNode()" << endl;
164     cout << "2. prependNode()" << endl;
165     cout << "3. insertNodeAfter()" << endl;
166     cout << "4. deleteNodeByKey()" << endl;
167     cout << "5. updateNodeByKey()" << endl;
168     cout << "6. print()" << endl;
169     cout << "7. Clear Screen" << endl << endl;
170 
171     cin >> option;
172     Node * n1 = new Node();
173     //Node n1;
174 
175     switch (option) {
```

```
176     case 0:
177       break;
178     case 1:
179       cout << "Append Node Operation \nEnter key & data of the Node to be Appended" << endl;
180       cin >> key1;
181       cin >> data1;
182       n1 -> key = key1;
183       n1 -> data = data1;
184       s.appendNode(n1);
185       //cout<<n1.key<<" = "<<n1.data<<endl;
186       break;
187 
188     case 2:
189       cout << "Prepend Node Operation \nEnter key & data of the Node to be Prepended" << endl;
190       cin >> key1;
191       cin >> data1;
192       n1 -> key = key1;
193       n1 -> data = data1;
194       s.prependNode(n1);
195       break;
196 
197     case 3:
198       cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New node: " << endl;
199       cin >> k1;
200       cout << "Enter key & data of the New Node first: " << endl;
201       cin >> key1;
202       cin >> data1;
203       n1 -> key = key1;
204       n1 -> data = data1;
205 
206       s.insertNodeAfter(k1, n1);
207       break;
208 
209     case 4:
210 
```

```
211         cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
212         cin >> k1;
213         s.deleteNodeByKey(k1);
214 
215         break;
216       case 5:
217         cout << "Update Node By Key Operation - \nEnter key & NEW data to be updated" << endl;
218         cin >> key1;
219         cin >> data1;
220         s.updateNodeByKey(key1, data1);
221 
222         break;
223       case 6:
224         s.printList();
225 
226         break;
227       case 7:
228         system("cls");
229         break;
230       default:
231         cout << "Enter Proper Option number " << endl;
232       }
233 
234     } while (option != 0);
235 
236     return 0;
237 }
238 
```

```cpp
// 4. Tree
#include <iostream>

struct Node {
int data;
Node* left;
Node* right;

Node(int data) {
    this->data = data;
    this->left = nullptr;
    this->right = nullptr;
}
};

class BinaryTree {
public:
Node* root;

BinaryTree() {
    root = nullptr;
}

void addNode(int data) {
    Node* newNode = new Node(data);

    if (root == nullptr) {
    root = newNode;
    } else {
```

```
30      Node* focusNode = root;
31      Node* parent;
32
33      while (true) {
34          parent = focusNode;
35
36          if (data < focusNode->data) {
37          focusNode = focusNode->left;
38          if (focusNode == nullptr) {
39              parent->left = newNode;
40              return;
41          }
42          } else {
43          focusNode = focusNode->right;
44          if (focusNode == nullptr) {
45              parent->right = newNode;
46              return;
47          }
48          }
49      }
50      }
51  }
52
53  void preOrderTraversal(Node* focusNode) {
54      if (focusNode != nullptr) {
55      std::cout << focusNode->data << " ";
56      preOrderTraversal(focusNode->left);
57      preOrderTraversal(focusNode->right);
58      }
```

```
59  }
60  };
61
62  int main() {
63  BinaryTree tree;
64
65  tree.addNode(50);
66  tree.addNode(25);
67  tree.addNode(75);
68  tree.addNode(12);
69  tree.addNode(37);
70  tree.addNode(43);
71  tree.addNode(30);
72
73  tree.preOrderTraversal(tree.root);
74
75  return 0;
76  }
77
```


THANK
YOU!