LAB MANUAL
Data Structures and Algorithms
CSC-201

University Institute of Information Technology
PMAS-Arid Agriculture University, Rawalpindi

DATA STRUCTURES AND ALGORITHMS (CSC-201)

Course Instructor: Dr. Rubina Ghazal
Office: Room 05
E-mail: rubinaghazal@uaar.edu.pk

Course Learning Outcomes (CLOs)

At the end of course the students will be able to:	**Domain**	**BT Level***
1. Implement various data structures and their algorithms, and apply them in implementing simple applications.	C	2,3
2. Analyze simple algorithms and determine their complexities.	C	4,5
3. Apply the knowledge of data structures to other application domains.	C	3
4. Design new data structures and algorithms to solve problems.	C	6
***Six BT levels** are: (1) knowledge, (2) comprehension, (3) application, (4) analysis, (5) synthesis, and (6) evaluation		

Course Objective

This course aims at teaching the students to write programs that not only are correct but also computation and space efficient and optimized for the intended use through appropriate structuring/organization of the related data. Students will learn the standard data structures such as linked lists, stacks, queues, trees, and graphs and the algorithms that manipulate them. Students will also be introduced to the concept of algorithm complexity analysis in order to make them realize the cost of the operations they perform on their data structures.

Outcomes

Upon the completion of Data Structures practical course, the student will be able to:

1. Design and analyze the time and space efficiency of the data structure.
2. Identity the appropriate data structure for given problem.
3. Understand the applications of data structures.
4. Choose the appropriate data structure and algorithm design method for a specified application.
5. Understand which algorithm or data structure to use in different scenarios.
6. Compare different implementations of data structures and to recognize the advantages and disadvantages of them.
7. Write complex applications using structured programming methods.

Lab-1
Fundamental OO Design and Implementation Practice

Q. No. 1: Implement the String class using C++ having following functions.

a) sLength() function
b) sCompare() function
c) sConcat() function
d) sCopy() function
e) sFind() function
f) sSubstring() function
g) Test all the functions by displaying all the results in the main function.

Lab-2

Implementation and Comparison of searching algorithm

Q. No. 1: Implement the linearSearch and BinarySearch using C++.

a) Implement the class LinearSearch.
b) Implement the class BinarySearch.
c) Compare and test both with respect to time on different inputs in main function and show that binary search is fast as compare to linear search.

Lab-3
Implementation and Comparison of sorting algorithm

Q. No. 1: Implement the different sorting algorithms using C++.

a) Implement the BubbleSort.
b) Implement the SelectionSort.
c) Implement the InsrtionSort.
d) Implement the MergeSort.
e) Implement the QuickSort.
f) Compare and test all above with respect to time on different inputs in main function and show the results.

Lab-4

Implementation of Static Stack.
Infix to Post fix Conversion and Evaluation

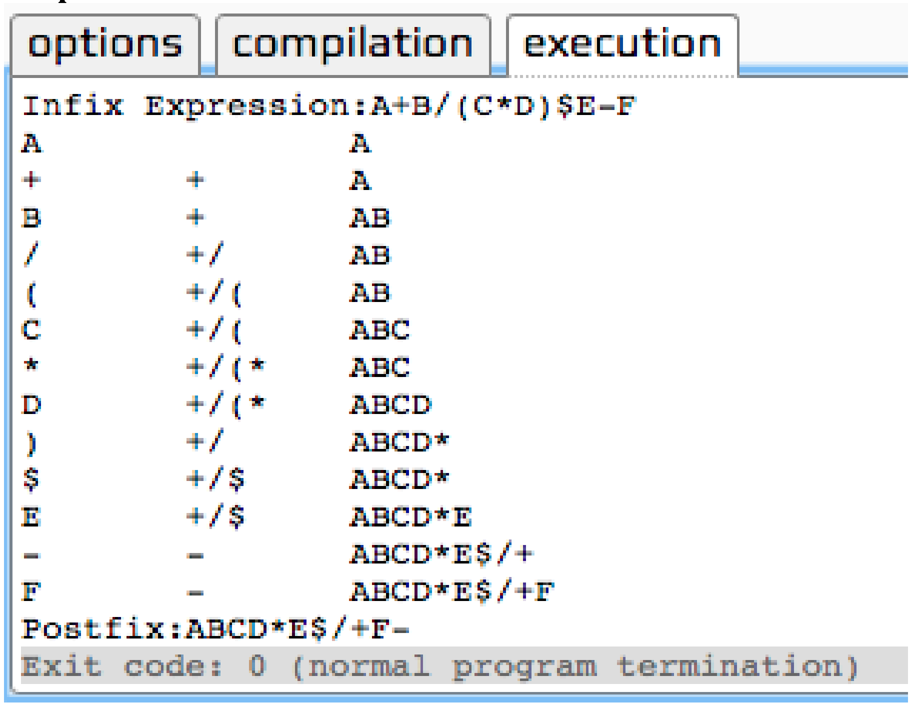
Q. No. 1: Convert the infix expression into postfix expression using C++ and array based stack.

a) Create a class stack having character array of size n, also it has push(), pop(), isFull(), isEmpty(), stackTop(), display() and precedence() functions.
b) Implement all the rules of infix to postfix conversion in the main() function and finally displays the postfix expression.
c) Test it on two different input expressions

Solution:

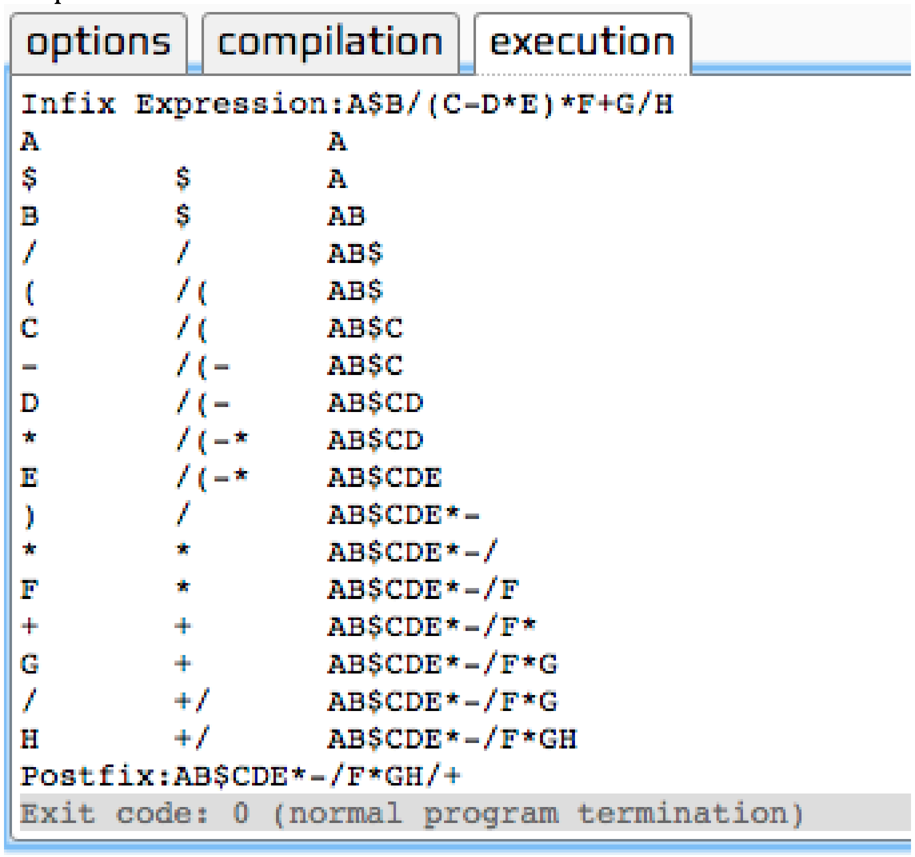
```
#include<iostream>
usingnamespacestd;
constint n=50;
class stack
{
        char items[n];
        int top;
        public:
          stack()
          {
                  top=-1;
          }
          void push(char x)
          {
                  top++;
                  items[top]=x;
          }
          char pop()
          {
                  char t=items[top];
                  top--;
                  return t;
          }
          boolisFull()
          {
                  if(top>=(n-1))
                          returntrue;
                  returnfalse;
          }
          boolisempty()
          {
                  if(top<0)
                          returntrue;
                  returnfalse;
          }
          charstacktop()
          {
                  return items[top];
          }
          void display(){
                  for(inti=0; i<=top;i++){
                          cout<<items[i];
                  }
          }
          bool precedence(char x){
                  if((x=='/'||x=='*')&&(stacktop()=='+'||stacktop()=='-'))
                          returntrue;
                  returnfalse;
          }
};
int main()
{
        stacks,t;
        inti=0;
        charsym;
        charexpr[]="A+B/(C*D)$E-F";
        //charexpr[]="A$B/(C-D*E)*F+G/H";
        while (!expr[i]=='\0')
        {
          sym=expr[i];
          if(sym>='A'&&sym<='Z' )
          {
                  s.push(sym);
          }
          if(sym==')'){
                  while(!s.isempty()){
                          if(t.stacktop()=='('){
                                  t.pop();
                                  break;
                          }
                          s.push(t.pop());
                  }
          }
          if(sym=='('||sym =='$'){
                  t.push(sym);
          }
          if(sym=='+'||sym=='-'||sym=='/'||sym=='*')
          {
                  while(!t.isempty())
                  {
                          if(t.stacktop()=='(')
                          {
                                  break;
                          }
                          else
                          {
                                  if(t.precedence(sym))
                                  {
                                          break;
                                  }
                                  else
                                          s.push(t.pop());
                          }
                  }
                  t.push(sym);
          }
          cout<<sym<<"\t";
          t.display();
          cout<<"\t";
          s.display();
          cout<<endl;
          i++;
        }
        while(!t.isempty())
        {
          s.push(t.pop());
        }
        cout<<"Postfix:";
          s.display();
        return 0;
}
```

Output: Test1



```
Infix Expression:A+B/(C*D)$E-F
A               A
+       +       A
B       +       AB
/       +/      AB
(       +/(     AB
C       +/(     ABC
*       +/(*    ABC
D       +/(*    ABCD
)       +/      ABCD*
$       +/$     ABCD*
E       +/$     ABCD*E
-       -       ABCD*E$/+
F       -       ABCD*E$/+F
Postfix:ABCD*E$/+F-
Exit code: 0 (normal program termination)
```

Output: Test2



```
Infix Expression:A$B/(C-D*E)*F+G/H
A               A
$       $       A
B       $       AB
/       /       AB$
(       /(      AB$
C       /(      AB$C
-       /(-     AB$C
D       /(-     AB$CD
*       /(-*    AB$CD
E       /(-*    AB$CDE
)       /       AB$CDE*-
*       *       AB$CDE*-/
F       *       AB$CDE*-/F
+       +       AB$CDE*-/F*
G       +       AB$CDE*-/F*G
/       +/      AB$CDE*-/F*G
H       +/      AB$CDE*-/F*GH
Postfix:AB$CDE*-/F*GH/+
Exit code: 0 (normal program termination)
```

Lab-5
Parenthesis matching validation implementation

Q. No. 1: Implement the parenthesis matching in mathematical expressions using C++.

a) Create a class stack having character array of size n, also it has push(), pop(), isFull(), isEmpty(), stackTop() and display()functions.

b) Implement all the rules of parenthesis matching validation in the main() function and finally displays either the expression is valid or not.

c) Test it on two different input expressions.

Lab-6
Implementation of queue operations, SJF& Double Ended Queue

Q. No. 1: Implement the Shortest Job First as Ascending Priority Queue using C++ and array based queue.

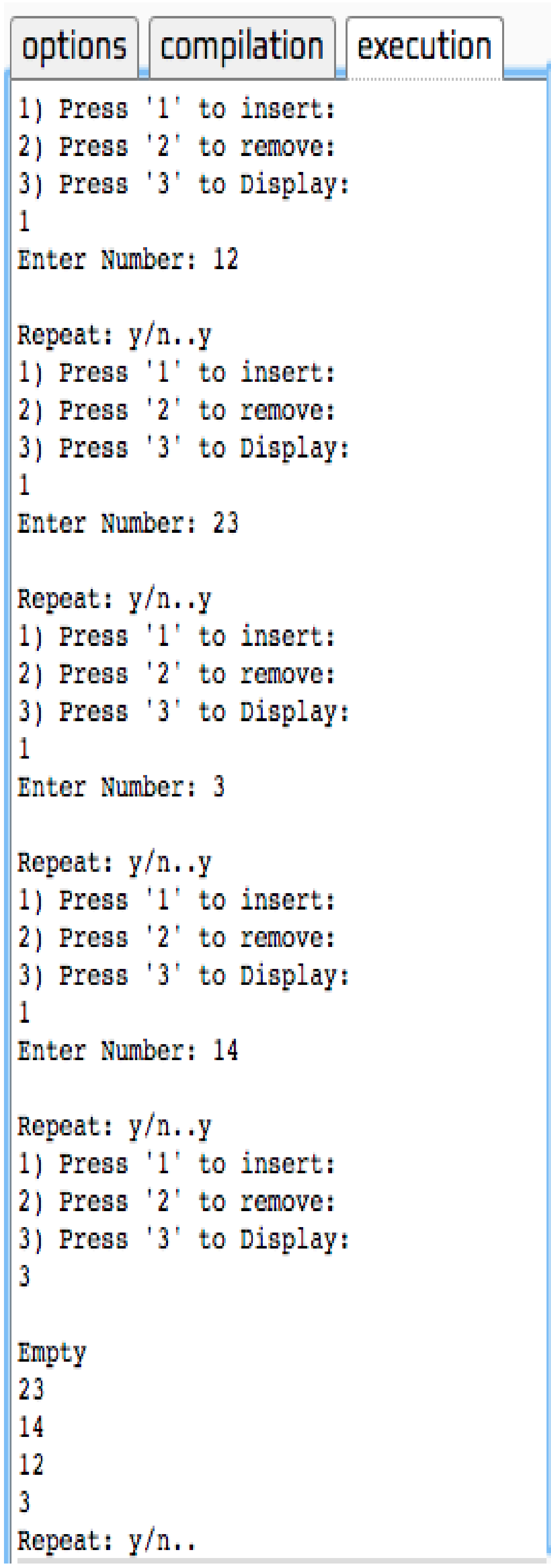
a) Create a class Queue having int array of size 5, also it has constructor to initialize front=0 and rear=-1. In addition to that it has inert(), ascendingPriorityInsert(), remove(), isFull(), isEmpty(), and display() functions.
b) Implement all the rules of ascending priority queue in apQueue() function .
c) Implement menu driven main function with three option of insert, remove and display.
d) Test it on two different input values.

Solution:

```
#include<iostream>
usingnamespacestd;
class Queue{
        int item[5];
        int front;
        int rear;
        public:
          //Constructor to initialize....
        Queue(){
                    front = 0;
                    rear = -1;
          }
         //INSERTION FUNCTION TO INSERT VALUES IN QUEUE....
          void insert(int x){
                    item[++rear]=x;
                    }
          boolisempty(){
                    if (front>rear)
                              returntrue;
                    returnfalse;
          }
         //INSERTION OF DATA IN QUEUE IN ASSENDING ORDER....
          voidapinsert(int x){
                    if(isempty()){
                              insert(x);
                    }
                    elseif (x>=item[rear])
                    {
                              insert(x);
                    }
                    else
                    {
                              Queue q2;
                              while(!isempty()){
                                        int t=remove();
```

```
                                if(x<t){
                                        q2.insert(x);
                                        q2.insert(t);
                                        break;
                                }
                                else
                                {
                                        q2.insert(t);
                                }
                        }
                        while(!isempty())
                                q2.insert(remove());

                        front=0;
                        rear=-1;
                        while(!q2.isempty()){
                                insert(q2.remove());
                        }

                }
        }
                int remove()
                {
                        return item[front++];
                }
                voiddislpay()
                {
                        for(int k = 4;k>rear;k--)
                        cout<<"\nEmpty";
                        for(inti = rear;i>=front;i--)
                        cout<<endl<<item[i];

                        for(int j = (front-1);j>=0;j--)
                        cout<<"\nEmpty";
                }
                boolisfull()
                {
                        if(rear>=4)
                                returntrue;
                                returnfalse;
                }
        };
        int main()                              // MAIN FUNCTION....
        {       Queue q1;

                char repeat= 'y';
               do{

               int input;
               cout<<"1) Press '1' to insert: "<<endl;
        cout<<"2) Press '2' to remove: "<<endl;
        cout<<"3) Press '3' to Display: "<<endl;
        cin>>input;
        system("cls");
        switch(input){
          case1:{
                    intnum;
                    cout<<"Enter Number: ";
                    cin>>num;
                    system("cls");
                    q1.apinsert(num);
                    break;
          }
          case2:{
                    if(q1.isempty())
                    cout<<"\nThe queue is already empty";
                    else
                    cout<<"\nThe removed value is:"<<q1.remove();
                    break;
          }
          case3:{
                    q1.dislpay();
                    break;
          }
          default:
                    cout<<"\nInvalid Option";
        }
        cout<<"\nRepeat: y/n..";
        cin>>repeat;
        system("cls");
        }while(repeat=='y');
        return0;
  }
```

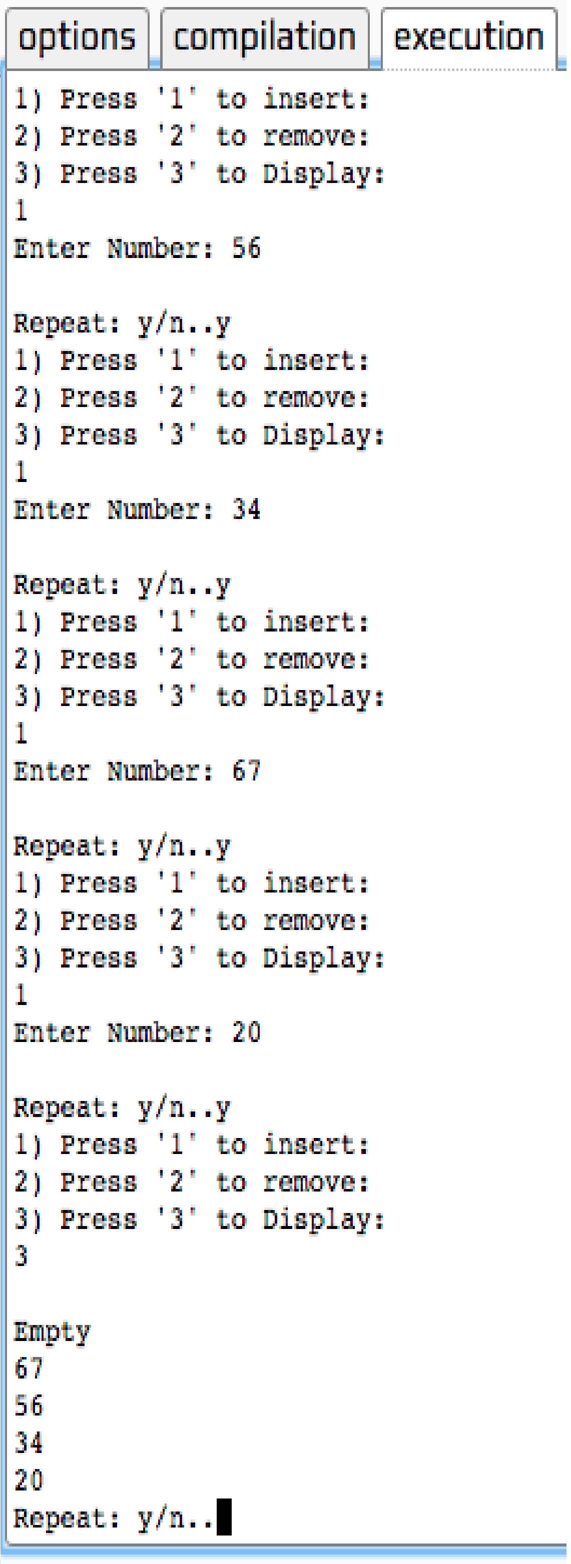
Output: Test1



options compilation execution

```
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 12

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 23

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 3

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 14

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
3

Empty
23
14
12
3
Repeat: y/n..
```

Output: Test2



options compilation execution

```
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 56

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 34

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 67

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
1
Enter Number: 20

Repeat: y/n..y
1) Press '1' to insert:
2) Press '2' to remove:
3) Press '3' to Display:
3

Empty
67
56
34
20
Repeat: y/n..
```

Q. No. 2: Implement the double ended input restricted array based Queue using C++.

a) Create a class DEIQueue having int array of size 5, also it has constructor to initialize front=0 and rear=-1. In addition to that it has inert(), ascendingPriorityInsert(), remove(), isFull(), isEmpty(), and display() functions.
b) Implement all the rules of ascending priority queue in apQueue() function .
c) Implement menu driven main function with three option of insert, remove and display.
d) Test it on two different input values.

Solution:

```
// Example program
#include<iostream>
#include<iomanip>
usingnamespacestd;
constint n=5;


classDEIQueue
{
private:
int items[n];
intrear,front;

public:
DEIQueue()
{   front=0;
rear=-1;
   }
void insert( int x)
   {
items[++rear]=x;

   }


int remove()
   {
return items[front++];
   }
intrearRemove()
   {
return items[rear--];
   }
boolisfull()
   {
if(rear>=(n-1))
returntrue;
returnfalse;

   }
```

```
boolisempty()
   {
if(rear<front)

returntrue;
returnfalse;

   }
void display()
   {
for(inti=(n-1);i>rear;i--)
cout<<"-------\n|Empty|\n";
for (int j=rear; j>=front;j--)
cout<<"-------\n|"<<setw(5)<<items[j]<<"|\n";
for(int k=(front-1);k>=0;k--)
cout<<"-------\n|Empty|\n";
cout<<"-------";
   }
};
int main() {
DEIQueue q;
int option,option2;
int v;
charch;
do
   {
cout<<"\nDouble ended Input restricted queue";
cout<<"\n-------------------------------------";
cout<<"\n press 1 for insert : ";
cout<<"\n press 2 for remove : ";
cout<<"\n press 3 for display: ";


cin>>option;

switch(option)
      {
case1:
if(q.isfull())
cout<<"\n Queue is already full\n";
else
            {
cout<<"\n Enter value\n";
cin>>v;
q.insert(v);
cout<<endl;
q.display();
            }
break;
case2:
if(q.isempty())
cout<<"\n Queue is already Empty\n";
else
{ cout<<"\nDouble ended Input restricted queue";
cout<<"\n------------------------------------";
cout<<"\n press 1 for front end remove : ";
cout<<"\n press 2 for rear end remove : ";
cin>>option2;

switch(option2)
                    {
case1:
q.remove(); break;
case2:
q.rearRemove(); break;
default: cout<<"\nInvalid Option";
                    }
q.display();
                }
break;

case3:
if(q.isempty())
cout<<"\n Queue is Empty\n";
else
q.display();
break;
default: cout<<"\nInvalid Option";

        }
cout<<"  \nDo You Want to continue? (Y/N)";
cin>>ch;
    }
while(ch=='y');

cout<<"\nThanks ";
return0;
}
```

Lab-7

Linklist implementation using primitive functions

Q. No. 1: Implement the linklist based Stack using C++.

a) Create a class StackLinklist having primitive functions of push() and pop().
b) In addition to primitive functions you have to add supporting functions of isEmpty(), stackTop() and display().
c) Implement menu driven main function with four option of push, pop, stackTop and display.
d) Test it on two different input values.

Q. No. 2: Implement the linklist based Queue using C++.

a) Create a class QueueLinklist having primitive functions of insert() and remove().
b) In addition to primitive functions you have to add supporting functions of isEmpty(), queueFront() and display().
c) Implement menu driven main function with four option of insert, remove, queueFront and display.
d) Test it on two different input values.

Lab-8

Queue modification as a Circular queue and Priority queue

Q. No. 1: Implement the linklist based Circular Queue using C++.

a) Create a class CQueueLinklist having primitive functions of insert() and remove().
b) In addition to primitive functions you have to add supporting functions of isEmpty(), queueFront() and display().
c) Implement menu driven main function with four option of insert, remove, queueFront and display.
d) Test it on two different input values.

Q. No. 2: Implement the linklist based Priority Queue using C++.

a) Create a class PQueueLinklist having primitive functions of insert(), remove() and priority_Insert().
b) Implement all the cases of priority insertion i.e. insertion at rear end, at front end and in between front and rear.
c) In addition to primitive functions you have to add supporting functions of isEmpty(), queueFront() and display().
d) Implement menu driven main function with four option of priorityInsert, remove, queueFront and display.
e) Test it on two different input values.

Lab-9
Implementation of doubly linklist

Q. No. 1: Implement the DoublyLinklist based Queue using C++.

a) Create a class DQueueLinklist having primitive functions of insert() and remove().
b) In addition to primitive functions you have to add supporting functions of isEmpty(), queueFront() and display().
c) Implement menu driven main function with four option of insert, remove, queueFront and display.
d) Test it on two different input values.

Q. No. 2: Implement the Doubly Circular linklist based Priority Queue using C++.

a) Create a class DCPQueueLinklist having primitive functions of insert(), remove() and priority_Insert().
b) Implement all the cases of priority insertion i.e. insertion at rear end, at front end and in between front and rear.
c) In addition to primitive functions you have to add supporting functions of isEmpty(), queueFront() and display().
d) Implement menu driven main function with four option of priorityInsert, remove, queueFront and display.
e) Test it on two different input values.

Lab-10
Binary Search Tree implementation

Q. No. 1: Implement the Binary search tree using C++.

a) Create a structure Node for data storage.
b) Create a class BST having primitive function of insert().
c) Also add preOrderDisplay(), inOrderDisplay() and postOrderDisplay() functions to see the true insertion in BST.
d) Implement all the cases of remove() from BST, i.e. removal of leaf node, removal of root node and removal of in between node.
e) Implement menu driven main function with four option of Insert, remove, BST_Root and display.
f) Test it on two different input values.

Lab-11

Max Heap and Heap Sort implementation

Q. No. 1: Implement the MaxHeap using C++.

a) Create a structure Node for data storage in a heap.

b) Create a class MaxHeap having primitive function of insert(), remove and heapify().

c) Also add display() function to see the true insertion in max heap.

d) Implement all the cases of remove() from Max heap, i.e. removal of leaf node, removal of root node and removal of in between node.

e) Implement menu driven main function with four option of Insert, remove, maxValue and display.

f) How we can use max heap for heap sort implementation.

g) Test it on two different input values.

Lab-12
Graph implementation of DFS and BFS search algorithms

Q. No. 1: Implement the Depth First Search (DFS) on graph using C++.

a) Store the graph in adjacency matrix.
b) Create a class DFS having all the required functions.
c) Test it on two different input values.

Q. No. 2: Implement the Breadth First Search (DFS) on graph using C++.

a) Store the graph in adjacency matrix.
b) Create a class BFS having all the required functions.
c) Test it on two different input values.

Lab-13

Implementation of shortest path (Warshal Floyd's algorithm)

Q. No. 1: Implement the Warshal's Algorithm on graph using C++ to find the path existence between any two pair of nodes.

g) Consider different cities of Pakistan as nodes and the road links between them as edges and Store the graph in adjacency matrix.
h) Create a class WarshalPath having all the required functions.
i) Add a function pathfinder() that input two cities and display either path exist between these two cities or not.
j) Test it on two different input values.

Q. No. 2: Implement the Floyd's Algorithm on graph using C++ to find the shortest path between any two pair of nodes.

a) Consider different cities of Pakistan as nodes and the distance between them as edges and Store the graph in adjacency matrix.
b) Create a class FloydShortestPath having all the required functions.
c) Add a function pathfinder() that input two cities and displays the shortest path if it exists between these two cities.
d) Test it on two different input values.