Data Structure & Algorithm Lab Manual

EE-213L

Fall 2026

University of Management & Technology

School *of* Engineering

Department of Electrical Engineering

Name : _____

Student ID : _____

Section : _____

Data Structure & Algorithm Lab

University of Management and Technology School of Engineering
Course Outline (Fall 2026)

Course Code: EE213L **Course Title: Data Structure and Algorithms Lab**

Program	BS Robotics and Intelligent Systems, Electrical Engineering
Credit Hours	1
Contact Hours	3
Knowledge Domain	Computing
Prerequisites	Nil
Resource Persons	M.Annas Marwat
Counseling Timing	See Office Window (SEN-310)
Email	muhammad.annas@umt.edu.pk

Chairman/Director signature_____

Dean signature_____

Date_____

‘

Data Structure & Algorithm Lab

Learning Objectives:

The students will be able to understand the fundamental concepts and program building blocks such as conditional and loops statements, functions in C++ Languages

Learning Methodology:

Course Outline Page 1 Lecture, interactive, participative

Grade Evaluation Criteria:

Following are the criteria for the distribution of marks to evaluate final grade in the semester.

Marks Evaluation	Marks
Lab Performance + Manuals	50
Open-Ended Lab	10
Final Viva + Performance	20+20
Total:	100

Recommended Text Books:

- Data Structures and Algorithms Made Easy by Narasimha Karumanchi
- Data Structures and Algorithms in C++ by Michael T. Goodrich, Roberto Tamassia, and David M. Mount

Reference Books:

- Object-Oriented Data Structures Using C++ Author: Michael Main & Walter Savitch

Course Learning Outcomes (CLOs):

CLO I: Reproduce the programs/codes using concepts of different data structures and their related operations on state-of-the-art compiler.

CLO 2: Explain the experimental results of different tasks appropriately.
CLO 3: Demonstrates responsible attitude during lab execution
CLO 4: Present the experimental results of different tasks and related outputs in appropriate way.

Data Structure & Algorithm Lab

Relation to EE Program Learning Outcomes (PLOs):

CLOS	Related PLOS	Levels of Learning	Teaching Methods	CLO Attainment checked in
CLO I	PLO 5	Psy-3	Instructions, Lab Work	Lab Sessions, Exam
CLO 2	PLO 10	Aff-2	Instructions, Lab work	Viva,Lab Sessions,Exam
CLO 3	PLO 4	Cog-2	Instructions, Lab Work	Viva,Lab Sessions,Exam
CLO 4	PLO 8	Aff-3	Instructions, Lab Work	Lab Sessions,Manuals

Calendar of Course contents to be covered during semester

Course Code: EE213L **Course Title: Data Structure and Algorithms Lab**

Week	Course Contents/Lab Experiments	CLOS
1	Introduction to Lists, References, and Strings in Python	
2	Getting the data inputs using to Static and Dynamic Lists with References	1,2,3,4
3	Implementation of Class and Constructors	1,2,3,4
4	Usage of Classes, Objects, Constructors and References to Objects	
5	Implementation of Linked Lists	
6	Circular and Double-Linked Lists	1,2,3,4
7	Implementation of Queues	1,2,3,4
8	Midterm Exam	
9	Implementation of Stacks	1,2,3,4
10	Searching Algorithms	1,2,3,4
11	Implementation of Binary Search Trees (Inorder Tree traversal, Searching)	1,2,3,4
12	usage of Binary Search Trees (Tree Size, Duplicate Node)	1,2,3,4
13	Usage of Binary Search Trees (Preorder and Postorder Tree traversal)	1,2,3,4

14	Usage of Binary Search Trees (Deletion Cases)	
15	Sorting Algorithms (Bubble Sort, Merge Sort, and Quick Sort)	1,2,3,4
16	Open Ended Lab	

Complex Engineering Problem/Activity:

Complex Engineering Problem Details	Included: No
Open Ended Lab	Included: Yes

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 4	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 1:

INTRODUCTION TO ARRAYS, POINTERS AND STRINGS

Objectives:

- To introduce the concept of pointer variables and reference variables.
- To introduce the concept of strings usage with pointers
- To present how pointers and references can be used with arrays and functions

Equipment:

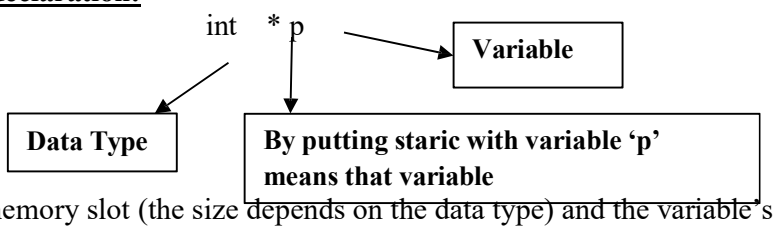
Personal Computer with CodeBlocks

Introduction:

A pointer is a variable that stores the address of another variable. There are two main operations when working with pointers in C++:

The Address Operator => &
The Value Operator => *

General format of pointer declaration:



Each variable is assigned a memory slot (the size depends on the data type) and the variable's data is stored there.

Pointers Arithmetic: Increments/Decrements through Pointer:

```
#include<iostream>
using namespace std;
void main()
{       int *p;
        int x;
        p=&x;
        cout<<&x<<endl;
        cout<<p<<endl;
        x=10;
        cout<<x<<endl;
        cout<<*p;
        cout<<"\n\n";
        system("pause");
}
#include<iostream>
using namespace std;
void main()
```

```
{       int *p,x;
        p=&x;
        *p=100;
        (*p)++;
        cout<<x<<endl;
        (*p)--;
        cout<<x<<endl;
        cout<<"\n\n";
        system ("pause");
}
```

Pointers and Arrays:

```
#include<iostream>
using namespace std;
void main()
{       int *p, j[10];
        p=j;  \\\ you can also write p=&j[0];
        cout<<&j[0]<<endl;
        cout<<p<<endl;
        p++;
        cout<<&j[1]<<endl;
        cout<<p<<endl;
        cout<<"\n\n";
        system ("pause");
}
#include<iostream>
using namespace std();
void main()
{       int *p[10],i[10],y;
        for (y=0;y<10;y++)
        {       i[y]=y;                 }
        for (y=0;y<10;y++)
        {       p[y]=&i[y];             }
        for (y=0;y<10;y++)
        {       cout<<*p[y]<<endl;      }
        cout<<"\n\n";
        system ("pause");
}
```

Data Structure & Algorithm Lab

Lab Assignments:

Q1: Write a program that will take ten values from the user. And the program should display the average of those ten values. This problem should be done through pointers.

Hint: Arrays and Pointers can be used.

Q2: Input and Display the 3 x 3 matrix through the 2D pointer. Write a function that will find out and then display the maximum and minimum values in the matrix.

Q3: Write a program to sort all elements of a 3x3 matrix by using pointers.

Lab Report

Data Structure & Algorithm Lab

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

Lab 2:

INTRODUCTION TO STATIC AND DYNAMIC ARRAYS WITH POINTERS

Objectives:

- To study static arrays using pointers
- To study dynamic arrays using pointers

Equipment:

Personal Computer with Microsoft Visual Studio 2013

Introduction:

Pointers can be used to do any operation involving array subscripting. Assume that integer array b[5] and integer pointer bptr have been defined. The pointer bptr can be set to point the array b[5]by the following statement:

bptr = b;

this statement is equivalent to taking the address of the first element of the array as follows:

bptr = &b[0];

array element b[3] can alternatively be referenced with the pointer expression:

*(bptr + 3);

The 3 in the above statement indicates the element at b[3] of the array b. The element 3 indicates the number of locations offset from the location number 0. This notation is referred to *pointer/offset* notation. Now consider the following expression:

*bptr + 3;

In this case, this expression will add 3 to the value of *bptr i.e. 3 is would be added to b[0] assuming bptr points to the beginning of the array. Just as the array element can be referenced with a pointer expression, the address

&b[3]

can be written with the pointer expression

bptr + 3

the array itself can be treated as a pointer and used in pointer arithmetic. For example, the expression

*(b + 3)

also refers to the array element b[3]. Pointers can be subscripted exactly as arrays can. For example, if bptrhas the value b, the expression

bptr[1]

refers to the array element b[1]. This is referred to as *pointer/subscript notation*. Now the following program will illustrate the use of pointers to traverse the elements of an array.

Pointers and Arrays:

Now the following program will illustrate the use of pointers to traverse the elements of an array.

```
#include<iostream>
using namespace std;
int main()
{     int *p, a[10], sum = 0;
p = a; // you can also write p=&j[0];
      cout<<&a[0]<<endl;
      cout<<p<<endl;
      p++;
      cout<<&a[1]<<endl;
      cout<<p<<endl;
      // initializing array a directly
      for(int i = 0; i < 10; i++)
{     cin>>a[i];  }

// summing the elements of the array
for(int i = 0; i < 10; i++)
{     sum = sum + a[i]; }
cout<<"sum of elements:"<<sum<<endl;
return 0;
}
```

Indirectly the pointer can also be used to assign and access elements of the array to which the pointer points. For this, the above program can be modified as follows:

```
#include <iostream>
using namespace std;
int main()
{     int *p, a[10], sum = 0;
      p = a;
      // initializing array a indirectly using pointer
      for(int i = 0; i < 10; i++)
{
      cin>>*(p + i); // this can be replaced by two statements
}                    // cin>>*p; p++;
// summing the elements of the array using pointer
for(int i = 0; i < 10; i++)
{
      sum = sum + *(p + i); // this can be replaced by two statements
}                           // sum = sum + *p; p++;
cout<<"sum of elements:"<<sum<<endl;
return 0;
}
```

Both of the above programming codes yield the same result.

There can be an array composed entirely of pointers in which every location of the array is a pointer itself. This means that every location of the pointer array will point to another memory location or another array location. This is shown in the following program:

```
#include<iostream.h>
int main()
{     int *p[10],a[10],y;
      for(y=0;y<10;y++)
      {     a[y] = y;    }
      for(y=0;y<10;y++)
      {     p[y]=&a[y]; }
      for(y=0;y<10;y++)
      {     cout<<*p[y]<<endl;       }
}
```

The above concept of 1 D array of pointers can be extended to implement a 2D array of pointers. Students should try to do this themselves.

Concept of Dynamic Arrays

Until now, in all programs, there was pre-defined memory declared for variables, having the size of all of them to be determined in the source code, before the execution of the program. But, what if we need a variable amount of memory that can only be determined during runtime? For example, in the case that we need some user input to determine the necessary amount of memory space. The answer is *dynamic memory*, for which C++ integrates the operators new and delete.

new operator

In order to request dynamic memory the operator new is used. new is followed by a data type. The number of elements for the dynamic memory is placed within brackets []. It returns a pointer to the beginning of the new block of memory allocated. Its form is:

pointer = new data-type
pointer = new data-type [number_of_elements]

The first expression is used to allocate memory to contain one single element. The second one is used to assign a block (an array) of elements, where number_of_elements is an integer value representing the amount of elements.

Example:

int *p;
p = new int [5];

In this case, the system dynamically assigns space for five elements of type int and returns a pointer
to the first element of the sequence, which is assigned to pointer p. Therefore, now, pointer p points to a valid block of memory with space for five elements of type int.

In order to assign the size of the array at run time, the ‘5’ in the above example should be replaced by a variable whose value will be taken by the user when the program is executed. Thus, enabling run-time memory allocation and creating dynamic memory. This is explained in the following program:

```
#include <iostream>
Using namespace std;
Int  main ()
{     int i,n;
      int * p;
      cout << "How many numbers would you like to type? ";
      cin >> i;
      p = new int[i];
      for (n=0; n<i; n++)
      {     cout << "Enter number: ";
            cin >> p[n];
      }
      cout << "You have entered: ";
      for (n=0; n<i; n++)
      {     cout << p[n] << ", ";    }
}
```

delete operator

Since the necessity of dynamic memory is usually limited to specific moments within a program, once it is no longer needed it should be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of the operator delete, whose format is:

delete pointer;
delete [] pointer;

Example:

In order to delete the pointer pointing to the dynamic memory in the above example, we will write the following statement at the end of the above program:

delete [] p;

The first expression should be used to delete memory allocated for a single element, and the second for memory allocated for arrays of elements. The value passed as argument to delete must be either a pointer to a memory block previously allocated with new, or a null pointer (in the case of a null pointer, deleteproduces no effect).

Lab Assignments

1. Following is the code for selection sorting. Write a program implement a user defined 3x3 array of integers. Then sort all elements of 3x3 array.
2. Following is the code for selection sorting. Write a program implement a user defined 3x3 array of integers using pointers. Then sort all elements of 3x3 array using pointers.
3. Following is the code for selection sorting. Implement this code to sort a dynamically created

1 D user defined integer array.

```
void selection(int array[], int size)
{
      int temp, j;
      int i = 0;
      while(i < size - 1)
      {
            j = i + 1;
            while(j < size)
            {
              if(array[i] > array[j])
              {
                  temp = array[i];
                  array[i] = array[j];
                  array[j] = temp;
              }
              j++;

            }
            i++;
       }
}
```

Lab Report

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 3:
INTRODUCTION TO CLASSES AND CONSTRUCTORS

Objectives:

- To study classes and objects in object oriented programming (OOP)

Data Structure & Algorithm Lab

- To study constructors and destructors. array of objects, pointer to objects and dynamic allocation of objects

Introduction:

This lab session introduces a brief idea of solving different problems using classes and constructors. Students are required to complete the programming during the lab and get it checked by the instructor. Besides, the students should also complete the lab assignments, given at the end of this manual, during the lab. Late lab submissions will be graded accordingly.

Use of Classes

Making a Class:

```
class vehicle
{
public:
        int doors; int
        seats;
};
```

This creates a class with the name "vehicle". This class has two variables that define the data for vehicles. Now this class is included in the program and objects are made from this class as follows:

```
#include <iostream> using
namespace std(); class vehicle
{public:
        int doors; int
        seats;
};

int main()
{
        vehicle taxi;                   // object instantiation
        taxi.doors = 4;                 // initializing objects of class vehicle taxi.seats = 3;          //
        object taxi

        cout<<"taxi can have "<<taxi.doors<<endl; cout <<"taxi has
        seats "<<taxi.seats<<endl;

}
```

All students are required to instantiate another object of class vehicle with any other vehicle name and define data for that vehicle. Also print the data corresponding to that vehicle.

Now modify the above program so that the program takes the input from the user to declare the number of doors and seats for the two vehicles defined in the program above.

Making Functions For Classes:

```
#include <iostream> using
namespace std(); class vehicle
{public:
        int  doors;   int
        seats; int fuel;
        int miles_per_litre;
        int range();      // function for class vehicle

};
int vehicle::range()                    // function definition
{       return fuel*miles_per_litre;  }

void main()
{        vehicle taxi;                  // object instantiation cout<<"range for taxi
         is: "<<taxi.range()<<endl; system("pause");
}
```

The range returned by the function can also be stored in some other variable like the following:

```
int main()
{        vehicle taxi;                  // object instantiation int range;
         range = taxi.range();  // variable "range" stores the range returned
                                                  // by the function taxi.range();
         cout<<"range for taxi is: "<<range<<endl;

}
```

Modify the above program in which two vehicle objects are created. Then:

- Enter the values of fuel and miles_per_litre for the two vehicle objects from the keyboard.
- Use the vehicle objects to call the same member function i.e. range(); and then calculate the values for the range of the two vehicles.
- In the main method, display the range returned by the two function calls.

Passing Parameters to the Member Functions:

```
#include <iostream> using
namespace std(); class vehicle
{public:
        int  doors;   int
        seats; int fuel;
```

```
        int miles_per_litre;
        int range(int f, int mpl);                    // function for class vehicle
};
int vehicle::range(int f, int mpl)   // function definition
{       fuel = f; miles_per_litre = mpl;
        return fuel*miles_per_litre;
}
int main()
{        vehicle taxi;              // object instantiation cout<<"range for taxi is:
        "<<taxi.range(10, 4)<<endl; system ("pause");
}
```

The main method for the above program can also be written as:

```
int main()
{        vehicle taxi;              // object instantiation int range;
        range = taxi.range(10, 4);                    // function call using object "taxi" cout<<"range for
        taxi is: "<<range<<endl;
        system ("pause");
}
```

Modify the above program to create two objects for vehicle class. Then:

- Declare two separate integer type variables in the main method. Name them "f" and "mpl".
- Now take user input for "f" and "mpl".
- Call the range function and pass the input values to the function parameters.
- Calculate range for both vehicle objects and display the range in the main method.

Public and Private Specifiers:

Consider the following code segment to understand the difference and use of private and public specifiers:

```
#include<iostream> using
namespace std(); class ratio
{private:
        int num; int
        den;

        int number;

public:
        int display_ratio(int n, int d);

};

int ratio:: display_ratio (int n, int d)
{       num = n;
        den = d;
        number = n/d; return
        number;
}
int main()
{       ratio r1; cout<<r1.display_ratio(4,2)<<endl;
```

```
        system (“pause”);
}
```

The class “ratio” can also be written as follows:

```
class ratio
{private:
        int num; int
        den; intnumber;

public:
        int display_ratio(int n, int d)
        {
                num = n;
                den = d;
                number = n/d; return
                number;
        }
};
```

Notice that if the member function is defined inside the class, class name is not used in the header. Now modify the above program such that:

- Enter the values for the numerator and denominator from the keyboard, declaring the variables as private.
- Define another member function for class “ratio” to print the ratio.
- In the main method, call the printing function using the object of class “ratio”.

```
#include<iostream> using
namespace std; class ratio
{private:
        intnum; int
        den;
public:
        void input()
        {       cout<<”enter value for numerator: ”; cin>>num;
                cout<<”enter value for denomenator: ”;
                cin>>den;
        }
        void print()
        {       cout<<"ratio is: "<<num<<"/"<<den<<endl;  }

};
int main()
{       ratio r1;               // creating object r1;
        r1.input(); // assigns values to num and den of r1 r1.print(); // print the ratio
        assigned to r1 system (“pause”);
}
```

Data Structure & Algorithm Lab

Lab Assignments

1. Make a class with the name "numbers". The class should have two member variables 'a' and 'b'. The class may look like:

```
class numbers
{public:
    int a, b; void
    input(); int add();
    int sub(); int
    mult();};
```

The member functions should be defined to:

- To initialize a numbers from the user (like the input() function above in example).
- To add two user defined numbers. Display the result.
- To multiply two user defined numbers. Display the result.
- To subtract two user defined numbers.

Call these functions in the main method using objects of class "numbers". Use these functions to create 1 object of class numbers. Now call all the arithmetic functions defined above to perform addition, subtraction and multiplication on this object.

2. Modify the above program to deal with complex numbers. The class 'complex' will have two member variables 'a' and 'b' to represent real and imaginary part of one complex number. The class may look like.

```
class complex
{public:
    int r, i, sum_r, sum_i,   … (you can add more variables); void input();
    void add();
    . . . (you can add more functions);};
```

Remember that the arithmetic operations on complex numbers are carried out as follows:

Addition: $(a + ib) + (x + iy) = (a + x) + i(b + y);$

Subtraction: $(a + ib) - (x + iy) = (a - x) + i(b - y);$

Multiplication: $(a + ib) * (x + iy) = a(x + iy) + ib(x + iy) = ax – by + i(ay + bx);$

Where *a, b, x, y* are real numbers and "*i*" stands for "*iota*".

Again call these functions in the main method for different objects of class "complex". One object of complex class will have one number with 2 parts (real and imaginary parts). In addition, implement separate function of display()to display one complex number in the complex number notation of "*a + ib"*.

Data Structure & Algorithm Lab

Lab Report

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 4:

USAGE OF CLASSES, OBJECTS, CONSTRUCTORS AND POINTERS TO OBJECTS

Objectives:

- To learn how to implement a list ADT using singly linked list
- To learn how to perform different operations like printing and searching within a linked list

Equipment:

Personal Computer with CodeBlocks

Introduction:

This lab session introduces a brief idea of solving different problems using classes, objects, constructors and Pointers to Objects.

Use of Constructor

Constructors are member functions with the same name as the class. A constructor is a special function that initializes the data members of a class. A class constructor function is called whenever an object of that class is created. Several constructors can be defined for one class. Consider the following code to understand the use of constructors.

```
#include<iostream>
Using namespace std;
class ratio
{private:
      int num;
      int den;

public:
      ratio()
      {     num = 0;
            den = 1;
      }
      void print()
      {     cout<<"ratio is:"<<num/den<<endl;    }
};

int main()
{     ratio r1;    // the constructor automatically assigns the values of 0
                   // and 1 to num and den, respectively
      r1.print(); // print the ratio to check what is the value in r1
      return 0;
}
```

Now consider the following program which has 2 constructors (of course of the same name as the class).

Similarly, any number of constructors can be defined having the same name but different parameter list.

```
#include<iostream>
using namespace std;
class ratio
{private:
      int num;
      int den;

public:
      ratio()
      {     num = 0;
            den = 1;
      }
      ratio(int n, int d)
      {     num = n;
            den = d;
      }
      void print()
      {     cout<<"ratio is: "<<num<<"/"<<den<<endl;  }
};

int main()
{     ratio r1;
      ratio r2(3,4);
      cout<<"displaying ratio 1:"<<endl;
      r1.print(); // print the ratio assigned to r1
      cout<<"displaying ratio 2:"<<endl;
      r2.print(); // print the ratio assigned to r2
      return 0;
}
```

There can be several constructors for one class, each initializing the different member parameters to different values. Remember that constructors do not have a return type. Declaring a return type for any constructor of a class will generate an error. Different constructors for a class can be differentiated from one another by having different parameter list. In this way, the objects of a class can be initialized to different initial values depending on how the object is created. For example, creating an object like ratio r1 will use the 1st constructor by default. Similarly, the object r2(3, 4) will be using the 2nd constructor. The member functions can be written in a different way as given in the following example:

```
#include<iostream>
Using namespace std;
class ratio
{private:
      int num;
      int den;

public:
      ratio();
      ratio(int n, int d);
      void print();
};
ratio::ratio()
{     num = 0;
```

```
	den = 1;
}
ratio::ratio(int n, int d)
{	num = n;
	den = d;
}
void ratio::print()
{	cout<<"ratio is: "<<num<<"/"<<den<<endl;
}

int main()
{	ratio r1;
	ratio r2(3,4);
	cout<<"displaying ratio 1:"<<endl;
	r1.print(); // print the ratio assigned to r1
	cout<<"displaying ratio 2:"<<endl;
	r2.print(); // print the ratio assigned to r2
	return 0;
}
```

The constructor function cannot be made private. This does not actually make sense to have a private constructor. Since private functions can only be used/called inside other functions of the class itself so it is not possible to use a private constructor to create objects of a class in the main method. A constructor may take values from the user as well, as in the following example:

```
ratio::ratio()

{
	cin>>num;
	cin>>den;
}
```

Pointer to Objects:

It is perfectly valid to create pointers that point to classes. We simply have to consider that once declared, a class becomes a valid data type, so we can use the class name as the type for the pointer. For example:

CRectangle *prect;

can act as a pointer to an object of class CRectangle.

In order to refer directly to a member (variables/functions) of an object pointed by a pointer we can use the arrow operator (->) of indirection. The following example creates an object a and a pointer *b of class CRectangle. The pointer b points to the object a. So in order to access the member variables or functions of class CRectangle we can use either the pointer or the object of class CRectangle.

```
#include <iostream.h>
class CRectangle
{	int width, height;
	public:
		void set_values (int, int);
		int area (void) {return (width * height);}
};
```

void CRectangle::set_values (int a, int b)

```
{      width = a;
       height = b;
}

int main ()
{      CRectangle a, *b;
       b = &a;
       a.set_values(1,2);
       cout << "a area: " << a.area() << endl;
       cout << "*b area: " << b->area() << endl;
       b->set_values(3,4);
       cout << "*b area: " << b->area() << endl;

       cout << "a area: " << a.area() << endl;
       return 0;
}
```

There can be overloaded functions (functions with same names but different parameter lists) to initialize class variables. The following example dynamically creates user defined number of objects and initializes the member variables using corresponding functions:

```
#include <iostream>
using namespace std;
class CRectangle
{      int width, height;
       public:
              void setvalues();
              void setvalues(int, int);
              int area (void) {return (width * height);}
};
void CRectangle::setvalues()
{      cout<<”enter value of width: ”;
       cin>>width;
       cout<<”enter value of height: ”;
       cin>>height;
}

void CRectangle::setvalues(int a, int b)
{      width = a;
       height = b;
}
int main ()
{      CRectangle *b, *d;
       int num, h, w;
       cout<<"enter number of objects you want to create: ";
       cin>>num;
       d = new CRectangle[num]; //Dynamic array of objects
       b = new CRectangle;
       for(int i = 0; i<num; i++)
       {      cout<<endl<<"enter values for object "<<i<<". . .
              "<<endl<<endl;
              d[i].setvalues();
       }
       for(i = 0; i<num; i++)
       {      cout <<endl<<"d["<<i<<"] area: " << d[i].area() << endl;    }
       cout<<"enter width for object b: ";
       cin>>w;
```

```
        cout<<"enter height for object b: "; cin>>h;
        b->setvalues(w,h);
        cout <<endl<<"*b area: " << b->area() << endl;

        delete[] d; delete b;
        return 0;
}
```

The above example allocates dynamic memory location for one object for the class, it would be

```
b = new CRectangle;
```

It creates an object of type CRectangle using new and returns a pointer to it called b.

Similarly, in order to allocate dynamic memory for an array of objects,

```
d = new CRectangle[2];
```

This instruction creates two pointer objects d[0], d[1].

Now in order to refer to the member functions and member data in a object pointed by a pointer, it can be either way

```
b->set_values (3,4);
or
(*b).set_values (3, 4);
```

the -> operator works with pointers to objects in just the same way that the dot (.) operator works with objects. But when we talk about dynamic array of pointer objects, for example d[0] and d[1] in the above example. We can simply access the member functions and member data with the help of dot (.) operator like d[0].set_values(3,4); You can also use constructors to initialize dynamically declared objects.

Summary:

Here is a summary on how to read some pointer and class operators (*, &, ., ->, [ ]) that appear in the above examples:

Expression can be read as

*x : pointed by x
&x : address of x
x.y : member y of object x
x->y : member y of object pointed by x
(*x).y : member y of object pointed by x (equivalent to the previous one)
x[0] : first object pointed by x
x[1] : second object pointed by x
x[n] : $(n+1)^{th}$ object pointed by x

Data Structure & Algorithm Lab

Lab Assignments

1. Make a class "rational". The user should enter numerator and denominator of the rational number separately. He/she should not enter the rational number. The member functions of the class should do the following:
   - **a.** Cubes the input rational number and displays the output using function in part a.
   - **b.** Multiply two rational numbers and displays the output using function in part a.
   - **c.** Adds two rational numbers and displays the result using function in part a.
   - **d.** Subtracts two rational numbers and displays the result using function in part a.
   - **e.** Create appropriate constructors for this class, keeping in mind that den = 0 will yield undefined result. In this way, it might be considered to initialize every new rational number object with den = 1 and num = 0. There may be other constructors, as described in the examples, to assign user defined values to the rational numbers.
   - **f.** Displays the rational number such that the number is printed in its lowest possible form. For example if the user enters 4 as denominator and 16 as numerator the member function prints 4 as output. If numerator is 7 and denominator are 3 the member function prints 7/3. Similarly if both the denominator and numerator are negative the rational number displayed should be positive. Implement separate conditions inside the function to check for these.

   This function can be made private, since we may not want everyone to use this function in the main method to display the rational number object itself. If this function is private it can be used/called in all the function described above to display the result of cube, multiplication, addition, subtraction, etc.

   NOTE: Before attempting this question, students should have a clear idea of addition, subtraction, multiplication of rational numbers.

2. Define a class STUDENT. You have to take Student ID, name, faculty, and marks in 4 subjects as input from the user. Then you have to calculate the Average of best 3 subjects' marks. You also need to find out the highest marks among those 4 subjects. In short, take the input from user, calculate, and display. You can start with the following class declaration as an example:

```
class STUDENT
{     public:
            char name[20], faculty[4];
            int ID, BEE_marks, FOP_marks, EDC_marks, DATA_marks;
            int input();
            int average_of_best_three();
};
```

   You can add more variable/functions to the above class as required.

   Note: You have to do this assignment with Pointer Objects.

Data Structure & Algorithm Lab

Lab Report

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 5:

INTRODUCTION TO LINKED LISTS

Objectives:

- To learn how to implement a list ADT using singly linked list
- To learn how to perform different operations like printing and searching within a linked list

Equipment:

Personal Computer with Code Blocks.

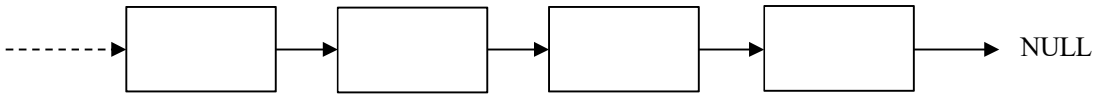
Introduction:

This lab session introduces an initial idea of dynamic data structure – linked lists. Students are required to complete the programming during the lab and get it checked by the instructor. Besides, the students should also complete the lab assignments, given at the end of this manual, during the lab.

Linked List:

Linked list is one of the fundamental data structures, and can be used to implement other data structures. It consists of a sequence of nodes, each containing arbitrary data fields and one or two links pointing to the next and/or previous nodes. The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk, allowing the list of items to be traversed in a different order. Besides a single node of the linked list can have data of multiple data types, unlike arrays where whole array is defined for one data type only.

Consider the following diagrammatic representation for a linked list.



Consider the following program for creating a linked list and printing the designed linked list.

```
#include <iostream>
using namespace std;
class node
{public:
      int data;
      node *next;
      void enter();
      void print();
};
void node::enter()
{     cout<<"Enter the Data: ";
      cin>>data;
}
void node::print()
```

```
{     cout<<"Data is: ";
      cout<<data;
      cout<<"\n";
}

int i;
node *start = NULL;
node *p;

void insertnodefront()
{     if(start == NULL)
      {     start = new node;
            start->enter();
            start->next = NULL;
            p = start;
      }
      else
      {     p->next = new node;
            p = p->next;
            p->enter();
            p->next=NULL;
      }
}
void printall()
{     p = start;
      while(p != NULL)
      {     p->print();
            p = p->next;
      }
}
```

Now consider the following program to search a node with specific data in the linked list.

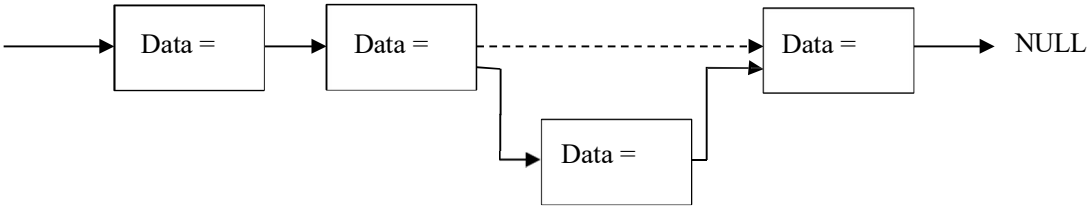
```
int search(int data)
{  p = start;
   while(p != NULL)
   {  if(p->data == data)
      {     return data;       }
      else
      {     p = p->next;       }
   }
   return -1;
}
```

Node Insertion:

Consider a linked list made up of several nodes. If it is required to insert a new node after a specific node in the linked list, that specific node should be first searched in the linked list. Following is a method to add a new node after a specific node in the linked list:

- Initialize a variable which stores the data of the node to be searched.
- Traverse the linked list (p = p->next inside a loop) from start to NULL and compare data of every node with the data to be searched (using an if condition).
- When that specific node is found, use references (pointers) to help insert the new node
- Initialize data to variables of the new node.
- Check if the link is connected to all the nodes (including the new node)

The following diagram will help you understand how to implement this function:



In the above example, a new node with data = 6 is inserted after the node with data = 3. Following the diagram above, once the node with data = 3 is found you need to change the references (pointers) appropriately to insert the new node in this list. The *next pointer of node with data = 3 should be modified to point to the new node and the *next pointer of the node with data = 6 should point to the next node in the list. Display the modified linked list after insertion of the new node to check if the new data is being displayed or not.

Node Deletion:

If it is required to delete a particular node from the linked list, there can be the following three possibilities:

- Deleting node from front of the list
- Deleting node from the end of the list
- Deleting node from in between the list

Deleting node from front of the list: This function can be implemented just by changing the node to which *start pointer points i.e. the *start pointer should not point to the first node rather it should point to the node next to the first node.

Deleting node from the end of the list: This function is implemented by changing the *next pointer of the second last node to NULL. In this way the last node will be excluded from the list.

Deleting node from in between: this function is quite similar to the "insert node in between" function. First, the node with a specific data (the node that is to be deleted) is searched in the list. Once that node is found, the reference (pointer) of the node previous to it should be modified to point to the node next to the node to be deleted.

Data Structure & Algorithm Lab

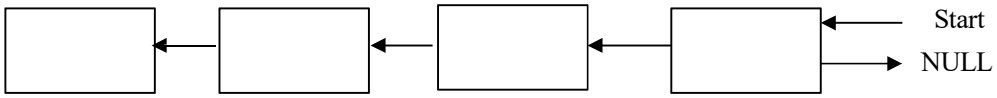
Lab Assignments

1. Use prog1.c to enter data in the linked list. Use the method of insertnodefront()to create a linked list with 5 nodes. Then use that printlist()method to print the linked list. Make all function calls in the main method.

2. Modify prog1.c to carry integer data and character data as well. This linked list will carry the data of students of one section. Insert user-defined number of nodes in the list. The integer data should represent the registration number of the student added in the linked list. Character strings in the nodes should specify the name of the student. Also modify the methods of void node::enter()and void node::print()to enter and print the character strings as well.

3. Modify the above program and use the method search(int data) to search for a student with specific registration number entered by the user.

4. All the nodes are inserted at the front of already inserted nodes in previous implementations of linked lists. Try implementing a method insertnodeback()that inserts a node behind an already inserted node. (hint: use another pointer node *q; to do that).



Inserting node from back

5. Create a linked list with 6 nodes (using insert node method from above), each node containing some integer data and a pointer (You can use the program in Question 1). Now implement the
   following methods:
   - **a)** Implement a method void insert_node_between(int data) to insert a new node after the node with specific data value (passed to the function in the data variable).
   - **b)** Implement a method void delete_node_end()to delete a node from the end of a linked list.
   - **c)** Implement a method void delete_node_start()to delete a node from the front of a linked list.
   - **d)** Implement a method void delete_node_between(int data)to delete a node with specific data passed as parameter.

6. Suppose you are assigned the job of keeping a record of the number plates of the cars parked outside a specific security building. It somehow, clicked in your mind that you once studied some sort of data structures- linked lists- that you think can be implemented here to collect the desired data. Implement the following:
   a) A class with the name ‘car’ that contains the number plate of the car and a pointer to point to the next car in the parking area. The number plate data would contain both letters and numerals. For this you will have to declare a character array.
   b) A method to insert a new car in the parking lot and to note down its number plate data. (modify the function insert_node())
   c) A method to delete data of a specific car when it leaves the parking lot. You have to search for the car parked before the one that just left. Or you can simply delete the car

by finding out its position in the parking lot. (Modify remove_node_between(int data) method for linked lists.)

d) Say if a new car is parked in the place emptied by the car that just left. Implement a method to insert this new car in the list in the place of the one that just left. Also note down the number plate data of the new car. You can also insert the new car at a specific unoccupied position by asking pos from the user. (modify the function insert_node_inbetween(int data))
e) At any time you may need to check if the parking lot is empty. (Implement function bool isEmpty()which returns 0 if the lot is empty and 1 otherwise)
f) Say the authorities ask you to search in the parking lot if a car with a specific number plate data is still parked there. Implement a method to fulfill their request (modify the function
void search (int data)).

You need to keep a count of the number of cars parked in the parking lot at any time. You may do this by assigning a variable to keep the count and displaying it whenever required.

Data Structure & Algorithm Lab

Lab Report

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 6:

INTRODUCTION TO CIRCULAR AND DOUBLY LINKED

LISTS

Objectives:

This lab session introduces the deletion operation in a tree to delete a leaf node, node with a single child and node with two children.

Equipment:

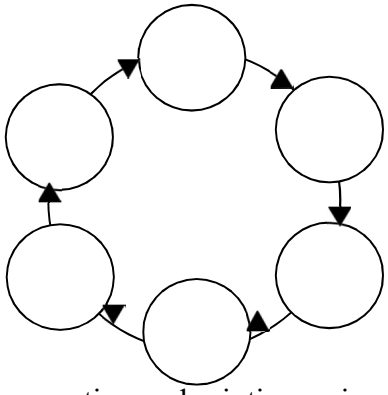
Personal Computer with Code Blocks.

Introduction:

This lab session introduces the idea of circular and doubly linked lists.

Circular Linked List

A circularly linked list is the same as a singly linked list but the last node of a circularly linked list does not point to NULL, rather it points to the first node in the linked list. A typical circularly linked list can be visualized as:



Consider the following program for creating and printing a circularly linked list;

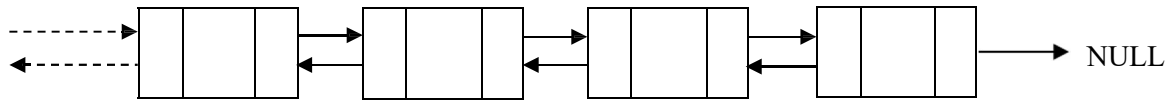
```
#include <iostream>
Using namespace std;
struct node
{     int data;
      node *next;
};
      node *start=NULL;
      node *p;
void insertnodefront()
{     if(start==NULL)
      {     start = new node;
            cin>>start->data;
            start->next = NULL;
            p = start;
      }
      else
      {     p->next = new node;
```

```
            p = p->next;
            cin>>p->data;
            p->next=start;
      }
}
void printlist()
{     p =start;
      cout<<"Data is: "<<p->data<<"\n";
      p=p->next;
      while(p!=start)
      {     cout<<"Data is: "<<p->data<<"\n";
            p=p->next;
      }
      cout<<"Data is: "<<p->data<<"\n";
      p=p->next;
}
```

There is another way of implementing the printlist function using do-while loop.

Doubly Linked List

A doubly linked list is the one in which there are two pointers in each node, one pointing in the forward direction to the next node in the list and the other pointing in the backward direction to the previous node in the list. Observe the structure defined for the node of a doubly linked list. It has two pointers, next and previous. A doubly linked list can be visualized as:




```
#include <iostream>
Using namespace std;
struct node
{     int data;
      node *next;
      node *prev;
};
      node *start=NULL;
      node *p;
      node *q;

void insertnodefront()
{     if(start==NULL)
      {     start = new node;
            cin>>start->data;
            start->next = NULL;
            p = start;
      }
      else
      {    q  =  new  node;
            cin>>q->data;
            p->next= q;
            q->prev=p;
```

```
            q->next=NULL;
            p=q;
      }
}
void printlist()
{     p =start;
      while(p!=NULL)
      {     cout<<"Data is: "<<p->data<<"\n";
            p=p->next;
      }}
```

LAB ASSIGNMENTS

1. Create a circularly linked list with 6 nodes. Print the circularly linked list. Use function from examples in the lab manual. Make all function calls in the main method.
2. Implement 3 different function to delete from start, end or in between the circular linked list. Call the functions in the main and display the circular linked list after each deletion.
3. Create a doubly linked list with 6 nodes. Print the doubly linked list. Use functions from examples in the lab manual. Make all function calls in the main method.
4. Implement a function void printreverse()to display the doubly linked list in reverse order. Call the function in the main method to display the list in reverse order.

Lab Report

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #							
	Implementation of	Code Organizati	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results		

Data Structure & Algorithm Lab

Areas Assessed			Lab Work	on				CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

INTRODUCTION TO QUEUES

LAB 7:

Objectives:

- To learn how to implement queue ADT using arrays and lists.
- To learn how to perform different operations like printing and searching in queues.
- To learn how to apply theses ADTs to solve real life problems.

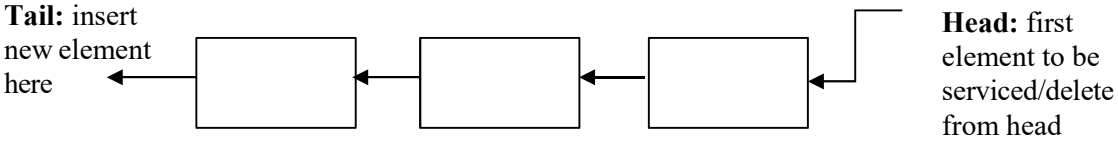
Equipment:

Personal Computer with Code Blocks.

Introduction:

A queue is a common data structure. A queue is very similar to a check-out line in the grocery store. The first person in the line is serviced first and other customers enter the line only at the end and wait to be serviced. Queue nodes are removed only from the head of the queue and are inserted only at the tail of the queue. Queues are therefore also known as **FIFO** (First In First Out) data structures. The fundamental operation on queues are:

- ***Enqueue:*** is equivalent to insert, inserts an element in the queue
- ***Dequeue:*** deletes the element inserted first



Applications:

Queues have many applications in computer systems.

1. Many computers have only a single processor, so entry for one user can be services at a time and others are placed in queue. Entries gradually advance.to the front of the queue as the entries from the front are serviced.
2. Queues also support print spooling. With a multiuser environment having only a single printe r, the
   print requests from various users are arranged in a queue with the first request to be serviced first
3. Queues also support print spooling. With a multiuser environment having only a single printe r,
   the print requests from various users are arranged in a queue with the first request to be serviced first.
   Information packets in computer networks also wait in a queue for being serviced.

Implementation of Queues:

Two popular implementations of queues are:
Using arrays

- Using pointers i.e. linked lists

Students will see how queues are implemented using these two methods. Further, they will come acrossthe advantages and drawbacks of each one of them.

Array Implementation of Queues:

The first method is the implementation of queues using arrays. The following program implements the functions for declaring the size of queue, checking if the queue is empty or full and adding elements to the queue.

```
#include <iostream>
Using namespace std;
class queue
{public:
      int array[20], top, max;
      void initial(int s);
      bool empty();
      bool full();
      void enqueue(int element);
      void dequeue();       // implement this method yourself
      void display();     // implement this method yourself
};
void queue::initial(int s)
{     max = s - 1;
      top = max;
}
bool queue::empty()
{     return top == max;       }
bool queue::full()
{     return top == -1;        }
void queue::enqueue(int element)
{     if(full())
      {     cout<<"Error: the Queue is full"<<endl;    }
      else
      {     array[top] = element;
            top--;
      }
}
```

In the lab assignments, you will implement the functions of dequeue and display (to display the queue).
There are following disadvantages of this implementation:

1. The array created will be of fixed size for one queue object.
2. One queue array will have elements of one data type only. In the above example, the queue will have the data of integer type only.

These issues are resolved when the queues are implemented with linked lists.

Linked List Implementation of Queues:

The second implementation of queue uses linked list. The following program illustrates enqueuer method to add elements to the queue.

```
#include <iostream>
Using namespace std;
class node
{public:
      int data;
      node *next;
};
node *start = NULL;
node *p;
void enqueue()
{     if(start == NULL)
      {     start = new node;
            cin>>start->data;
            start->next = NULL;
            p = start;
      }
      else
      {     p = start;
            while(p->next != NULL)
            { p = p->next; }
            p->next = new node;
            p = p->next;
            cin>>p->data;
            p->next = NULL;
      }
}
```

In the lab assignments, you will have to implement several other methods related to queues.

Lab Assignments

1. Using prog1.c, do the following:
   - **a.** Implement a function display() to display the queue.
   - **b.** In the main method, create an object q1 for queue class.
   - **c.** In the main method, use the queue object q1 created in part b to call the function initial(), pass the size of the array you want to create to this function.
   - **d.** In the main method, use the object q1 to call the enqueue() function to initialize the locations of the queue array. You will have to call this function as many times as is the size of the array created. Fill the queue with user defined elements.
   - **e.** In the main method,call the display() function implemented in part a and display the queue from head to tail.
   - **f.** Implement dequeue()method to remove elements from the front of the queue.
   - **g.** In the main method, call thedequeue()method 3 times to remove first 3 elements from the queue.
   - **h.** Implement a “while loop” in the main method to dequeue all elements from the queue.
   - **i.** In the main method, calldisplay()function to display the queue. The display function should approprietly display that the queue is empty.

2. Use prog2.c to create a queue of 6 elements. Then:

   - **a.** Use enqueue() method to create the queue and fill the it with user defined elements.

b. Implement display() function for queues and display the queue.
c. Implement dequeue()method to remove elements from the front of the queue
d. Use dequeue()method to remove first 3 elements from the queue.
e. Use display() function again to display the edited queue.
f. Implement a function isEmpty() which checks if the queue is empty or not.
g. Call the isEmpty()function in the main method.
h. Implement a "while loop" in the main method to dequeue() all the elements.
i. Now repeat step g.
j. Then display() the queue again. This should now appropriately display that the list is empty.

Use the queue data structure to implement a customer shopping queue at a grocery store. For each customer being added to the queue call the enqueue() function. Each customer should carry a name and the number of items he shopped for which should be displayed using display() function. At the store counter a separate function should calculate their bill based on the number of items and display it. Use dequeue() function to remove the customer after being billed. Call other functions of queues as required in the main method.

Lab Report

Data Structure & Algorithm Lab

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Lab #										
Areas Assessed	Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
	(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)	Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)	Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				

Data Structure & Algorithm Lab

very Good (75 to 100%) Maximum Marks			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
	Student ID	Student Name	2	1	2	3	2	5	4	1	10

LAB 8:

INTRODUCTION TO STACKS

Objectives:

- To learn how to implement stack ADT using arrays and lists.
- To learn how to perform different operations like printing and searching in stacks.
- To learn how to apply theses ADTs to solve real life problems.

Equipment:

Personal Computer with Code Blocks.

Introduction:

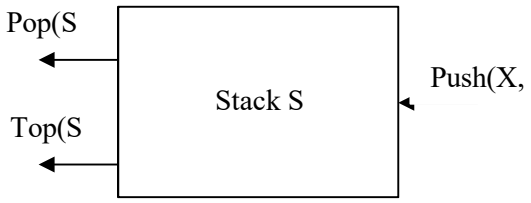
This lab session introduces an initial idea of dynamic data structure – stacks.

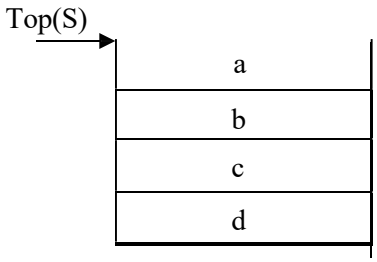
Stacks

A stack is a list with the restriction that the insertion and deletion can be performed in only one position, namely, the end of the list, called the top. Stacks are, therefore, also known as LIFO (last in, first out) lists. The fundamental operations on stacks are:

- *Push*: is equivalent to insert, inserts an element in the stack
- *Pop*: deletes the most recently inserted element
- *Top*: or peek: examines the most recently element before performing a pop

A *pop* or *top* on an empty stack is generally considered an error in stacks. On the other hand, running out of space when performing a *push* operation is an implementation of error.





This signifies that pushes are input operations and pops and tops are output operations. The second figure shows some elements in the stack but the element that is visible is the top most only.

Implementation of Stacks:

A stack is a list; any list implementation can also implement a stack. Two popular implementations of stacks are:

- Using pointers i.e. linked lists
- Using arrays

Students will see how stacks are implemented using these two methods. Further, they will come across the advantages and drawbacks of each one of them.

Array Implementation of Stacks:

There can be an array implementation of stacks. The methods for creating and printing stacks can also be implemented with the help of arrays.

```
#include <iostream>
Using namespace std;
class stack
{public:
      int array[20];
      int top element, max;

      void intial(int a);
      bool empty();
      bool full();
      void push(int element);
      void pop();        // implement this method yourself
      int top();         // implement this method yourself
      void display();    // implement this method yourself
};
void stack::intial(int a)
{     top_element = -1;
      max = a - 1;
}
bool stack::empty()
{     return top_element == -1;      }
bool stack::full()
{     return top_element == max;     }
void stack::push(int element)
{     if(full())
      {     cout<<"Error: the Stack is full"<<endl;    }
      else
      {     top_element++;
            array[top_element] = element;
      }
}
```

Linked List Implementation of Stacks:

The first implementation of stack uses linked list. Push is performed by inserting an element at the front of the list. Pop is performed by deleting an element from the front of the list. The top operation merely examines the element in the front of the list, returning its value. Sometimes pop and top operations are combined into one.

```
#include <iostream>
Using namespace std;
class node
{
Public:
      int data;
      node *next;
```

```
};

node *start = NULL;
node *p;

// push method to insert an element in the stack

void push(int element)
{
      if(start == NULL)
      {
            start = new node;
            start->data = element;
            start->next = NULL;
            p = start;
      }
      else
      {
            p = new node;
            p->next = start;
            p->data = element;
            start = p;
      }
}
```

Lab Assignments:

1. Using prog1.c, do the following:
   a. Implement a function display() to display the stack.
   b. In the main method, create an object s1 for stack class.
   c. In the main method, use the stack object s1 created in part b to call the function initial(), pass the size of the array you want to create to this function.
   d. In the main method, use the object s1 to call the push() function to initialize the locations of the stack array. You will have to call this function as many times as is the size of the array created. Fill the stack with user defined elements.
   e. In the main method, call the display() function implemented in part a and display the stack from top to bottom of the stack.
   f. Implement pop() method to remove elements from the top of the stack.
   g. In the main method, call the pop() method 3 times to remove first 3 elements from the stack.
   h. Implement top() method to display the element at the top of the stack only.
   i. In the main method, call the top() method to check the element at the top of the stack.
   j. Implement a “while loop” in the main method. Inside the while loop: display the element at the top of the stack and then remove the top most element. Use the while loop to remove all elements from the stack.
   k. In the main method, call display()function to display the stack. The display function should appropriately display that the stack is empty.
2. Use prog2.c to create a stack of 6 elements. Then:

Data Structure & Algorithm Lab

a. Use push() method to create the stack and fill the it with user defined elements.
b. Implement display() function for stack and display the queue.
c. Implement pop() method to remove elements from the top of the stack
d. Use pop() method to remove first 3 elements from the stack.
e. Use display() function again to display the edited stack.
f. Implement top() method to display the element at the top of the stack only.
g. In the main method, call the top() method to check the element at the top of the stack.
h. Implement a function isEmpty() which checks if the stack is empty or not.
i. Call the isEmpty() function in the main method.
j. Implement a "while loop" in the main method. Inside the while loop: display the element at the top of the stack and then remove the top most element. Use the while loop to remove all elements from the stack.
k. Now repeat step i.
l. Then display() the stack again. This should now appropriately display that the list is empty.

m. Implement a parenthesis checker using stacks. Parenthesis checker checks if the parenthesis in a user defined algebraic expression are balanced or not. Display appropriate message in either case. (HINT: Combine the methods of top() and pop() for this program)

n. Create a stack to determine if a user defined string is a palindrome (i.e. a string that is spelled identically backward and forward). The program should ignore spaces and punctuation marks.

Lab Report

Data Structure & Algorithm Lab

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Data Structure & Algorithm Lab

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 9:

INTRODUCTION TO BINARY SEARCH TREES

Objectives:

Data Structure & Algorithm Lab

- To learn the concept of Binary Search Trees

Equipment:

Personal Computer with Code Blocks.

Introduction:

This lab session introduces various algorithms about trees, binary trees – their construction, deletion, traversal and searching.

Trees:

For large amounts of input, the linear access time of the linked list is not desired. There is a simple data structure called trees for which the running time of most operations is O(log N) on average.

Types of Trees:

There are various types of trees based on the structure that are generally classified as N-ary trees and these are sub-divided into binary trees, red-black trees, AVL trees, etc. All of these have different algorithms for creation, traversal, insertion, deletion, etc. We will particularly focus on binary trees only.

Binary Tree:

By definition a binary tree, T is a set of nodes with the following properties:

1. Either the set is empty, $T = \Phi$; or
2. The set consists of a root r, and exactly two distinct binary trees T_r and T_l : $T = \{r, T_r, T_l\}$, where T_r and T_l are right subtree and left subtree of tree T, respectively.

Structure of a Tree:

Following are the terms related to a tree:

Root:

Root node or head node is the top node in a tree.

Child nodes:

All the nodes below the top root node are called child nodes.

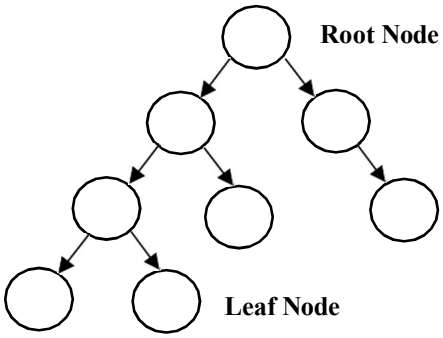
Parent nodes:

Nodes above child nodes are called parent nodes. Parent nodes can be children to other parent nodes.

Leaf nodes:
Nodes having no children are called leaf nodes.

Degree of node:
Degree of node is the number of children a node can have.

Degree of tree:
Number of levels in a tree except the root level is called degree of a tree or tree height. A full binary tree (a binary tree in which every node has two children except the leaf nodes) of height $h \geq 0$ has 2^h leaves.



binary tree with node degree or size of 8 and tree degree or height of 3

Binary Search Tree (BST):

A binary search tree is a binary tree with the following properties:
1. The left sub-tree of a node contains only nodes with values less than the node value
2. The right sub-tree of a node contains only nodes with values greater than the node value.
3. Both the right and left sub-trees must also be binary search trees.
4. There must be no duplicate nodes.

Creation of a Binary Search Tree:

Like other data structures, trees are created by inserting nodes using linked lists. The insert node method for creating trees is given as follows:

CODE:
```
#include <iostream>
using namespace std;

class node {
public:
   int data;
   node *left;
   node *right;
};
```

```
// Global variables (consider moving these inside main or a class)
node *start = NULL;

void insertnodefront(int data) {
    node *p = start;
    node *q = NULL;

    if (start == NULL) {
        start = new node;
        start->data = data;
        start->left = NULL;
        start->right = NULL;
    }
    else {
        // Traverse to find the correct leaf position
        while (p != NULL) {
            q = p; // q follows p to keep track of the parent
            if (p->data > data) {
                p = p->left;
            } else {
                p = p->right;
            }
        }

        // p is now NULL, q is the parent node where we attach the new node
        if (q->data > data) {
            q->left = new node;
            q->left->data = data;
            q->left->left = NULL;
            q->left->right = NULL;
        } else {
            q->right = new node;
            q->right->data = data;
            q->right->left = NULL;
            q->right->right = NULL;
        }
    }
}

int main() {
    // Example usage
    insertnodefront(10);
    insertnodefront(5);
    insertnodefront(15);
```

```
    cout << "Nodes inserted successfully." << endl;
    return 0;
}
```

Lab Assignments

1. Use insertnodefront() method of trees to create a level 4 tree. Print the corresponding tree using inorder() method.
2. The function to create a binary search tree, implemented above, does not satisfy property 4 of binary search tree. Modify prog1.cpp such that node with duplicate value is not inserted in a binary search tree. Use the modified insertnodefront() method to create a tree and use inorder() method to display the tree.
3. Implement a Boolean-valued method isEmpty() that returns true if the root of a tree is an empty tree. Call this method in main to check if it works.
4. Implement a Boolean-valued method isLeaf(node *p) that returns true if a given node is a leaf node. Call this method in another function to check if it works.
5. Implement the method binarysearchtree() to search the tree for a node with specific data. Call this method in main. Make proper use of the value returned from the function binarysearchtree().
6. Implement the method deletenode() to delete a node from the binary tree. Your delete function should be able to delete nodes with one child only. Call this function in main method and delete a user-defined node.
7. Implement a method treesize() that returns the number of nodes of the binary tree. Call that method in another function to find out the size of any given binary tree.

Lab Report

Data Structure & Algorithm Lab

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Data Structure & Algorithm Lab

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 10:

INTRODUCTION TO BINARY SEARCH TREES

Objectives:

- To learn the concept of Binary Search Trees

Equipment:

Personal Computer with Code Blocks.

Introduction:

This lab session introduces various algorithms about trees, binary trees – their construction, deletion, traversal and searching.

Trees:

For large amounts of input, the linear access time of the linked list is not desired. There is a simple data structure called trees for which the running time of most operations is O(log N) on average.

Types of Trees:

There are various types of trees based on the structure that are generally classified as N-ary trees and these are sub-divided into binary trees, red-black trees, AVL trees, etc. All of these have different algorithms for creation, traversal, insertion, deletion, etc. We will particularly focus on binary trees only.

Binary Tree:

By definition a binary tree, T is a set of nodes with the following properties:

3. Either the set is empty, $T = \Phi$; or
4. The set consists of a root r, and exactly two distinct binary trees T_r and T_l : $T = \{r, T_r, T_l\}$, where T_r and T_l are right subtree and left subtree of tree T, respectively.

Structure of a Tree:

Following are the terms related to a tree:

Root:

Root node or head node is the top node in a tree.

Child nodes:

All the nodes below the top root node are called child nodes.

Parent nodes:

Nodes above child nodes are called parent nodes. Parent nodes can be children to other parent nodes.

Leaf nodes:

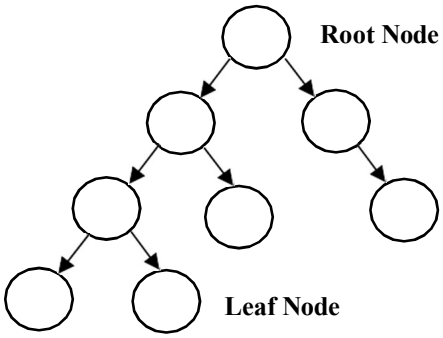
Nodes having no children are called leaf nodes.

Degree of node:

Degree of node is the number of children a node can have.

Degree of tree:

Number of levels in a tree except the root level is called degree of a tree or tree height. A full binary tree (a binary tree in which every node has two children except the leaf nodes) of height $h \geq 0$ has 2^h leaves.



binary tree with node degree or size of 8 and tree degree or height of 3

Binary Search Tree (BST):

A binary search tree is a binary tree with the following properties:

5. The left sub-tree of a node contains only nodes with values less than the node value
6. The right sub-tree of a node contains only nodes with values greater than the node value.
7. Both the right and left sub-trees must also be binary search trees.
8. There must be no duplicate nodes.

Tree Traversal:

There are different methods of traversing a tree which are:

1. Breadth first traversal
2. Depth first traversal
   a. Inorder traversal: left sub-tree, root node and then right sub-tree
   b. Preorder traversal: root node, left sub-tree and then right sub-tree
   c. Postorder traversal: left sub-tree, right sub-tree and then root node

In this lab session, only inorder tree traversal will be implemented. The method is as follows:

CODE:

```
#include <iostream>
using namespace std;

struct node {
    int data;
    node *left;
    node *right;
};

node *start = NULL; // Global root pointer

// Function to insert a node into the BST
void insertnodefront(int data) {
    node *p = start;
    node *q = NULL;

    if (start == NULL) {
        start = new node;
        start->data = data;
        start->left = NULL;
        start->right = NULL;
    }
    else {
        while (p != NULL) {
            q = p;
            if (data < p->data)
                p = p->left;
            else
                p = p->right;
        }

        node *newNode = new node;
        newNode->data = data;
        newNode->left = NULL;
        newNode->right = NULL;

        if (data < q->data)
            q->left = newNode;
        else
```

```
            q->right = newNode;
    }
}
```

Lab Assignments

8. Use insertnodefront() method of trees to create a level 4 tree. Print the corresponding tree using inorder() method.
9. The function to create a binary search tree, implemented above, does not satisfy property 4 of binary search tree. Modify prog1.cpp such that node with duplicate value is not inserted in a binary search tree. Use the modified insertnodefront() method to create a tree and use inorder() method to display the tree.
10. Implement a Boolean-valued method isEmpty() that returns true if the root of a tree is an empty tree. Call this method in main to check if it works.
11. Implement a Boolean-valued method isLeaf(node *p) that returns true if a given node is a leaf node. Call this method in another function to check if it works.
12. Implement the method binarysearchtree() to search the tree for a node with specific data. Call this method in main. Make proper use of the value returned from the function binarysearchtree().
13. Implement the method deletenode() to delete a node from the binary tree. Your delete function should be able to delete nodes with one child only. Call this function in main method and delete a user-defined node.
14. Implement a method treesize() that returns the number of nodes of the binary tree. Call that method in another function to find out the size of any given binary tree.

Lab Report

Data Structure & Algorithm Lab

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Data Structure & Algorithm Lab

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

LAB 11:

USAGE OF BINARY SEARCH TREES

Objectives:

- To learn how to delete data from BST

Equipment:

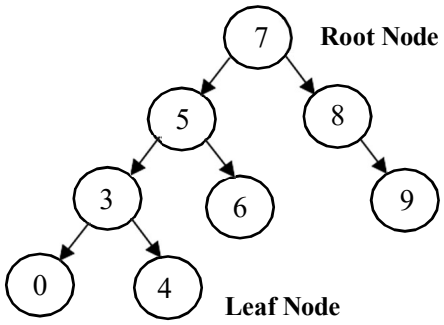
Personal Computer with Code Blocks.

Introduction:

This lab session introduces various algorithms about trees, binary trees – their construction, deletion, traversal and searching.

Deletion In Trees

For large amounts of input, the linear access time of linked list is not desired. There is a simple data structure called trees for which the running time of most operations is O(log N) on average.



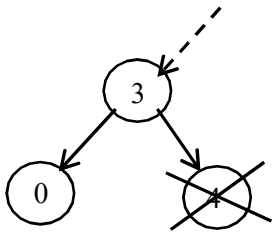
Deletion in binary search trees can be categorized into three different cases depending on the node to delete. The cases are:

1. Deleting leaf node
2. Deleting node with one child
3. Deleting node with two child

In all 3 cases, the first step will be to search the node to delete. For all cases, there will be 2 pointers. One pointer will be on the node to delete and he second pointer will be on the parent node of the node to delete. Different cases for node deletion are discussed below:

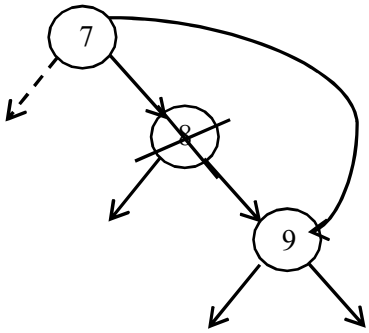
Case 1: deleting leaf node:

In this case, let's consider that the node to delete has the value 4. There will be one pointer on node 4 and another pointer on node 3(parent of 4). Your code should now set the right side of node 3 to NULL and then delete node 4.



Case 2: deleting node with one child:

In this case, the node to be deleted is node 8. One pointer should be on the node to delete. First determine if the node to delete has one child node on the right or on the left. In the case of node 8, the left child is NULL. Another pointer should be made to point to the parent of node 8. In order to delete node 8, the right of parent node 7 (since the right of parent node is to be deleted) should be pointed to the child of the node 8 to be deleted. It should be checked if the node 8 has a child on right or on left. If node 8 has child on right as in this case, the parent of node 8 should connect to right child of node 8. Otherwise, parent node should connect to left child of node 8. The same process should be implemented to delete a node with one child on left and NULL on right.



Case 3: deleting node with two child

In this case, the node to be deleted will be node 5. One pointer should be on node 5. In this case, the node to the right of node 5 should be set in place of node 5. If there is a right child to node 6 then value 5 will be replaced by 6 and new node 5 will point to right of node 6. In case if the right child of node 5 has a child node to the left then the left most node should be copied in place of node 5. These cases will be explained in detail in the class.

Tree Traversal:

There are different methods of traversing a tree which are:

1. Breadth first traversal
2. Depth first traversal
   a. Inorder traversal: left sub-tree, root node and then right sub-tree
   b. Preorder traversal: root node, left sub-tree and then right sub-tree
   c. Postorder traversal: left sub-tree, right sub-tree and then root node

In this lab session, preorder and postorder tree traversal will be implemented.

Data Structure & Algorithm Lab

Lab Assignments

1. Use insertnodefront() method of trees to create a tree with user defined number of nodes. Print the corresponding tree using preorder()and postorder()tree traversal method.
2. Implement a function tree_delete() to remove a user-defined node from the binary search tree. After deleting, use inorder() method to display the modified tree.
3. Implement the inorder() method of traversal iteratively. Call it in main method to check if it works correctly
4. Implement a recursive function tree_height(node *p) to determine the height of binary search tree created in question 1. Remember that the height of an empty tree is -1.
5. Implement a function child_nodes(node *p, int d) to determine and display the data of left and right child nodes of the node with data d passed as parameter. The function should first search for the node with data d in the BST. If the specific node is present in the BST, display the data of its right and left child nodes. If any of the right or left child nodes are NULL display accordingly.

Lab Report

Data Structure & Algorithm Lab

EE213L Data Structure and Algorithms Lab Evaluation Rubric

Data Structure & Algorithm Lab

Lab #											
Areas Assessed			Implementation of Lab Work	Code Organization	Ability to Debug Codes	Analyze the outcome of the codes	Illustration of experimental results	CLO wise Total			Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 4)				
poor (0 to 40%)			Code was not properly functional	Code was not properly organized	Wasn't able to debug the code	No analysis on the results obtained in the lab	Wasn't able to summarize the experiment	CLO 1	CLO 2	CLO 3	
Satisfactory(40 to 75%)			Code was somewhat functional	Code was somewhat neat	Debug the code with assistance	Reasonable interpretation of results obtained	Was able to summarize some of the experiments				
very Good (75 to 100%)			Code was fully functional	Code was properly organized	Debugged Code successfully without assistance	Thorough understanding of the results obtained in the lab	Thorough understanding of every aspect of experiments conducted				
Maximum Marks			2	1	2	3	2	5	4	1	10
	Student ID	Student Name									

Data Structure & Algorithm Lab

Data Structure And Algorithm
Open Ended Lab

Open-Ended Task

Problem Statement (Maximum Time: 3 hours)

Design and implement an efficient C++ program for a general store management system.

The program must have the option to **display list of items** along with **price and available quantities**. A user should be able to:

- Add items to the cart,
- Remove items from the cart and
- Display detailed bill (sorted by item name).

Please note that whenever a user **adds an item to the cart** the **quantity of that item reduces from the store.**

Submission Instructions

- Code of this lab on A4 page and attach a snapshot on word file.
- Output printout must be attached.
- You have to submit this Open-Ended lab on **June 16, 2026**. During the lab.
- Time allowed: **1 week**
- **Individual work only**
- Late submissions will not be entertained.