A

INDUSTRIAL TRAINING REPORT

On

"Data Structure and Object-Oriented Programming"

Submitted in partial fulfillment for the award of degree of

S.Y. B. Tech.

in

INFORMATION TECHNOLOGY

Submitted By

Krishna Ranjit Wable (21141216)

Under the Guidance of

Prof. R. S. Mawale

Government College of Engineering, Karad

(An Autonomous Institute of Government of Maharashtra)

Academic Year 2022-23

Acknowledgment

Apart from individual efforts, the success of Industrial Training depends largely on the encouragement and guidelines of many others. I take this opportunity to express our gratitude to the people who have been instrumental throughout the training period.

It is our privilege to express our gratitude towards our industry mentor *(Ms. Priya Chinchansure)* and academic guide *(Prof. R. S. Mawale sir)* for their valuable guidance, encouragement, inspiration and whole-hearted cooperation throughout the project work. I thank him for being a motivation through all our highs and importantly, our lows.

I deeply express our sincere thanks to our Head of Department Dr. S. J. Wagh for encouraging and allowing us to present the skills gained and work done during training period on " Data Structure And Object Oriented Programming In CPP " and providing us with the necessary facilities to enable us to fulfill our training requirements as best as possible. I take this opportunity to thank all faculty members and staff of Department of Information Technology, who have directly or indirectly helped our project.

I pay our respects to honorable Principal Dr. A. T. Pise for their encouragement. My thanks and appreciation also go to our family and friends, who have been a source of encouragement and inspiration throughout the industrial training.

Government College of Engineering, Karad

(An Autonomous Institute of Government of Maharashtra)

Department of Information Technology

CERTIFICATE

This is to certify that the Industrial Training entitled "Data Structure and Object Oriented Programming" has been carried out by Krishna Ranjit Wable (21141216) of Second Year B.Tech I.T. class under the guidance of Prof. R. S. Mawale during the academic year 2022-23 (Sem-III).

Prof. R.S. Mawale
(Guide)

Dr. S. J. Wagh
Head
Information Technology
Department

INFORMATION ABOUT THE COMPANY

IFS India Mercantile Pvt. Ltd. Pune provides the best Data Structure and Object Oriented Programming Training in online mode. IFS India Mercantile Pvt. Ltd. Pune labs are equipped with latest software so that students can get 100% practical training. They specialize in creating and improving products and services according to the current market needs. They also assist clients with market exploration and discovery for business development. Alongside this, they guide clients in the implementation of marketing changes to make a discernible business difference. Furthermore, they keep them updated on the latest marketing trends.

Company Name	IFS India Mercantile Pvt. Ltd. Pune
RoC	Pune
Directors	Santoshasai Palakurthy, Rajeswari Kondeti
Involved in	Data Structure and Oops
Class of Company	Private
Date of Incorporation	31st March, 2009
Contact Details	Email ID: bhatewara@gmail.com

TABLE CONTENT

MY INDUSTRIAL TRAINING EXPERIENCE

As a part of online training conducted by 'IFS India Mercantile Pvt. Ltd. Pune' it was on "Data Structure and Object Oriented Programming". I got a list of tasks to accomplish every day. Initially, we were been trained on the basic concepts of C++ before getting into the "Object Oriented Programming" concept, and then we progressed with "Data Structure". Every day we were asked to perform operations on different datasets on the basis of what we have learned. Once we are done with the task given the solution was discussed and made sure that it is conveyed in the easiest way that is possible.

A data structure is a group of data elements that provides the easiest way to store and perform different actions on the data of the computer. A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data Structure Concepts: Linear Data Structure (Array, Queue, Stack, Linked List) and Non-Linear Data Structure (Trees and Graphs).

As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function. Object Oriented Programming Concepts: Class, Objects, Data Abstraction, Encapsulation, Inheritance, Polymorphism, etc.

The tasks were based on topics covered they include

CHAPTER – I

1. PROGRAM EXECUTION AND DATA TYPES:

1.1 Program Execution in System:

Program is a set of instructions that you give to a computer so that it will do a particular task. The program is executed in the following manner, firstly the program is saved in it's extension form. Then it is been preprocessed or complied by the compiler and the code is converted into object code. Then the libray files and linkers combinedly converts object code to executable code (Fig. 1).

1.2 Types Of Languages:

a. **Low Level**: Faster for processing but development time is high.

e.g. Machine Language, Assembely Language.

b. **Mid Level**: Moderate speed for execution and takes moderate time for development.

e.g. c,c++ (this are close to hardware).

c. **High Level**: Low speed of execution but high time for development.

e.g. python, Java.

1.2.1 Languages baesd upon compiler, interpter and hybrid:

a. Compiler based: C , C++

b. Interpeter based: Python, JavaScript.

c. Hybrid based: Java

- Interpreter execute program line by line and stops execution when error is identified, while complier executes program in one go and collects all error at one time and shows to user.

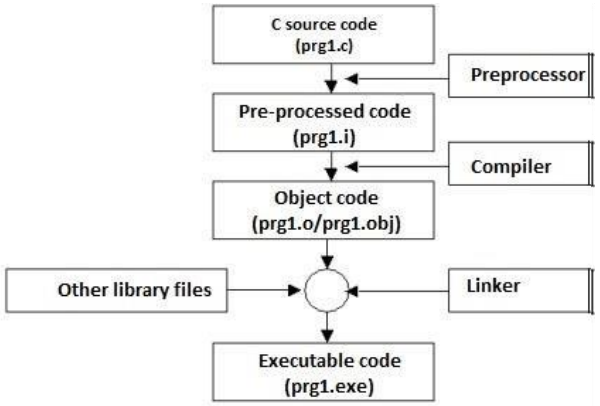


Fig. Program Execution

1.3 Programming Paradigm:

a. Monolithic Programming: If, we write an entire program in a single function that is in main function then, you call it as a monolithic type of programming.

b. **Modular / Procedural Programming**: If the program is divided into number of functional parts, then we use to call it as modular programming.

c. **Object Oriented Programming**: Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behaviour.

d. **Aspect Oriented / Component Assembly Programming**: In computing, aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.

1.4 Data and Data types:

Data is information such as facts and numbers used to analyse something or make decisions. Computer data is information in a form that can be processed by a computer. A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational, or logical operations can be applied to it without causing an error (Fig.2).

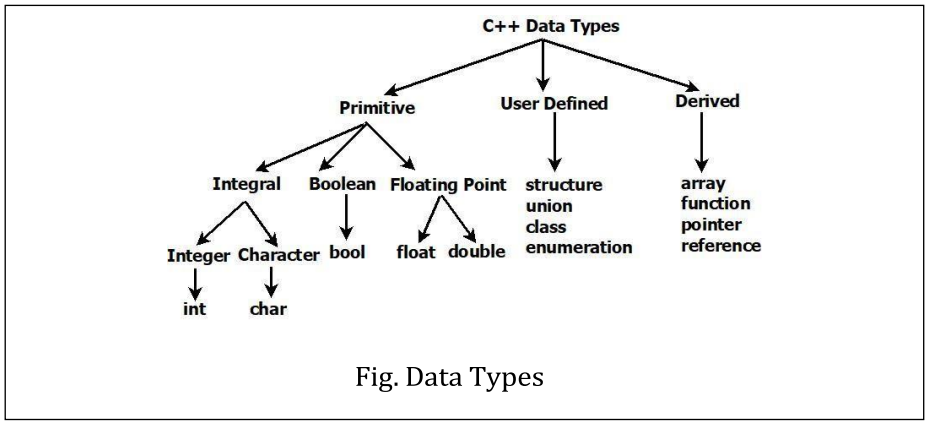


Fig. Data Types

1.4.1 Typecasting:

Typecasting is the process in which the compiler automatically converts one data type in a program to another one.

1.4.2 Enum and Typedef:

Enumerated type (enumeration) is a user-defined data type which can be assigned some limited values. These values are defined by the programmer at the time of declaring the enumerated type.

2. Conditional statement and declaration:

2.1 Conditional Statement:

a. **if else**: Decision making in programming is similar to decision making in real life. In decision making, a piece of code is executed when the given condition is fulfilled. Sometimes these are also termed as the Control flow statements.

b. **switch case:** Switch case statement evaluates a given expression and based on the evaluated value (matching a certain condition); it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions (cases). Switch case statements follow a selection-control mechanism and allow a value to change control of execution.

2.2 Looping Statements:

a. **while:** While Loop is used in situations where we do not know the exact number of iterations of the loop before-hand. The loop execution is terminated on the basis of the test condition.

b. **do while**: Like while the do-while loop execution is also terminated on the basis of a test condition. The main difference between a do-while loop and a while loop is in the do-while loop the condition is tested at the end of the loop body, i.e. do-while loop is exit controlled whereas the other two loops are entry-controlled loops.

c. **for**: A for loop is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

d. **for each:** 'for each' loop is used to iterate over the elements of a containers (array, vectors etc.) quickly without performing initialization, testing and increment/decrement. The working of 'for each' loops is to do something for every element rather than doing something n times.

2.3 Dynamic Allocation:

Dynamic allocation or run-time allocation is the allocation in which memory is allocated dynamically. In this type of allocation, the exact size of the variable is not known in advance. Pointers play a major role in dynamic memory allocation.

Stack - All the variables that are declared inside any function take memory from the stack. Heap - It is unused memory in the program that is generally used for dynamic memory allocation.

3. Array and Search:

3.1 Array:

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together.

3.1.1 One Dimensional Array:

A One-Dimensional Array is the simplest form of an Array in which the elements are stored linearly and can be accessed individually by specifying the index value of each element stored in the array (Fig.3).

3.1.1 Multi-Dimensional Array:

A multi-dimensional array can be termed as an array of arrays that stores homogeneous data in tabular form. Data in multidimensional arrays are stored in row-major order (Fig.4).

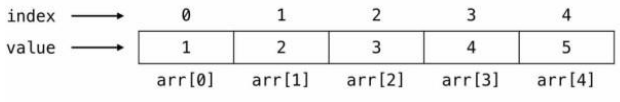


Fig 3. 1D Array

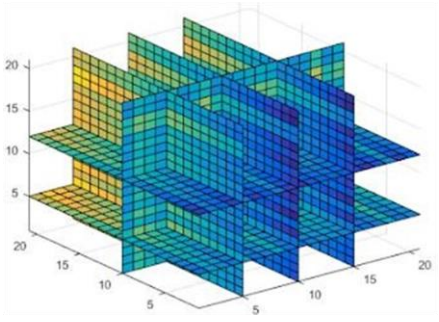


Fig 4 . 2D Arrays

3.2 Linear Search:

Linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search**.**

3.3 Binary Search:

Binary Search is a method to find the required element in a sorted array by repeatedly halving the array and searching in the half. This method is done by starting with the whole array. Then it is halved.

4. Pointer and Functions:

4.1 Pointer:

Pointers are used to store the address of variables or a memory location. This variable can be of any data type i.e. int, char, function, array, or any other pointer. The size of the pointer depends on the architecture (Fig. 5).

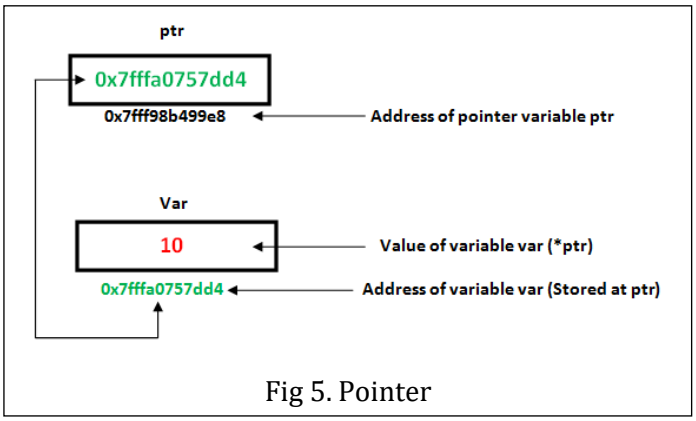


Fig 5. Pointer

4.2 Function:

A function is a set of statements that take inputs, do some specific computation, and produce output. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

4.2.1 Default argument:

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn't provide a value for the argument. In case any value is passed, the default value is overridden.

4.2.2 Recursive Function:

A function that calls itself is known as a recursive function

4.2.3 Passing Parameters:

a. **Passing by value**: In this value is passed to the function, i.e. formal argument contains values.

b. **Passing by reference**: In this we pass the reference to the function, i.e. formal an argument contains reference address of actual argument.

c. **Passing by address**: In this address is passed to the function, i.e. formal an argument contains pointers which points to actual argument.

4.2.4 Returning Type:

a. **Returning by value**: This function returns the value.

b. **Returning by reference**: This function returns the reference of the specified argument.

c. **Returning by address**: This function returns the address.

4.3 References:

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting '&' in the declaration. It does not take any memory space.

CHAPTER - II

1. Basic structure of Object Oriented Programming (OOPs):

1.1 Abstraction:

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

1.2 Encapsulation:

Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

2. Class and Constructor:

2.1 Class:

A class is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

- When we write function inside class we call it as **Method**.
- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create **objects**.

2.2 Constructor:

Constructor is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor has the same name as the class or structure. It constructs the values i.e. provides data for the object which is why it is known as **constructors**.

2.2.1 Default constructor:

Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

2.2.2 Non-parameterised constructor:

Non-Parametric Methods use the flexible number of parameters to build the model. Non-Parametric Methods requires much more data than Parametric Methods.

2.2.3 Parameterised constructor:

It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

2.2.4 Copy constructor:

A copy constructor is a member function that initializes an object using another object of the same class. In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously, is known as a copy constructor.

2.2.5 Deep Copy Constructor:

In Deep copy, an object is created by copying data of all variables, and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to explicitly define the copy constructor and assign dynamic memory as well, if required. Also, it is required to dynamically allocate memory to the variables in the other constructors, as well.

3. Function in class:

3.1 Types of function on class:

3.1.1 Function overloading:

Writing a function within a same function, with same name but different number of parameters.

3.1.2 Function Overriding:

It is the redefinition of base class function in its derived class with same signature i.e. return type and parameters.

3.1.3 Function template:

The duplication of function by changing its data type is known as templating.

3.1.4 Inline Function:

Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the compiler at compile time. Inline function may increase efficiency if it is small.

3.2 Scope Resolution:

Scope resolution operator is '::'. It is used to access a global variable when there is a local variable with same name, to define a function outside a class, to access a class's static variables, in case of multiple Inheritance (If same variable name exists in two ancestor classes, we can use scope resolution operator to distinguish) and refer to a class inside another class (If a class exists inside another class we can use the nesting class to refer the nested class using the scope resolution operator).

3.3 Operator Overloading:

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator without changing its original meaning.

4. Inheritance:

The capability of a class to derive properties and characteristics from another class is called Inheritance (Fig.7).

4.1 Modes of Inheritance:

There are 3 modes of inheritance:

1. Public Mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

2. Protected Mode: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

3. Private Mode: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

4.2 Types of Inheritance:

Types of inheritance are:

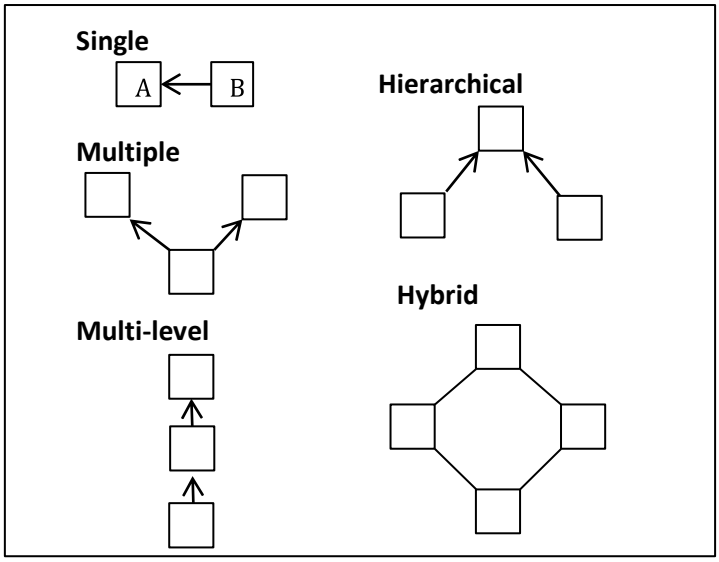


Fig 2.2 Types of Inheritance

4.2.1 Single Inheritance:

In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.

4.2.2 Multiple Inheritance:

Multiple Inheritance is a feature of C++ where a class can inherit from more than one class i.e. one subclass is inherited from more than one base class.

4.2.3 Multilevel Inheritance:

In this type of inheritance, a derived class is created from another derived class.

4.2.4 Hierarchical Inheritance:

In this type of inheritance, more than one subclass is inherited from a single base class i.e. more than one derived class is created from a single base class.

4.2.5 Hybrid (Virtual) Inheritance:

Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

4.3 IsA and HasA:

IsA is used for inheriting properties only, while HasA inherits using object of other class.

5. Polymorphism:

The word polymorphism means having many forms.

5.1 Types of polymorphism:

5.1.1 Compile Time Polymorphism:

This type of polymorphism is achieved by function overloading or operator overloading.

- Run time is the time period where the executable code is running.
- Errors can be detected only after the execution of the program.
- Errors that occur during the execution of a program are called run-time errors. Run time errors aren't detected by the compiler.

5.1.2 Run Time Polymorphism:

This type of polymorphism is achieved by Function Overriding. Late binding and dynamic polymorphism are other names for runtime

polymorphism. The function call is resolved at runtime in runtime polymorphism. In contrast, with compile time polymorphism, the compiler determines which function call to bind to the object after deducing it at runtime.

- Compile time is the time period where the code typed is converted into executable code.
- Errors are detected before the execution of the program.
- Errors that occur during compile time are called compile-time errors. Two types of compile-time errors area:

a. semantics error occurs when the statements aren't meaningful to the compiler.

b. Syntax error occurs when the programmer doesn't follow the syntax.

6. More about class and Exception handling:

6.1 Friend Function:

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be a member of another class or a global function.

6.2 Static Member of Class:

Static data members are class members that are declared using static keywords. A static member has certain special characteristics. These are only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created, it is initialized before any object of this class is being created, even before main starts and it is visible only within the class, but its lifetime is the entire program.

6.3 Nested / Inner Class:

A nested class is a class which is declared in another enclosing class. A nested class is a member and as such has the same access rights as any other member. The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.

6.4 Exception Handling:

Exceptions are runtime anomalies or abnormal conditions that a program encounters during its execution.

Exception handling consists of three keywords: try, throw and catch.

The try statement allows you to define a block of code to be tested for errors while it is being executed. The throw keyword throws an exception when a problem is detected, which lets us create a custom error. The catch statement allows you to define a block of code to be executed if an error occurs in the try block.

6.5 Pre-processor Derivatives / Macros:

A macro is a piece of code in a program that is replaced by the value of the macro. Macro is defined by #define directive. Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro. Macro definitions need not be terminated by a semi-colon.

7. Destructor and I/O stream:

7.1 Destructor:

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed. Characteristics of destructor frees the memory, free the resource, used to delocalisation of resource.

7.2 Virtual Destructors:

A virtual destructor is used to free up the memory space allocated by the derived class object or instance while deleting instances of the derived class using a base class pointer object. A base or parent class destructor use the virtual keyword that ensures both base class and the derived class destructor will be called at run time, but it called the derived class first and then base class to release the space occupied by both destructors.

7.3 I/O stream:

7.1 istream:

The class istream is used for input. This class is responsible for handling input stream. It provides number of function for handling chars, strings and objects such as get, getline, read, ignore, putback etc.

7.2 **ostream**:

The class ostream is used for the output. This class is responsible for handling output stream. It provides number of function for handling chars, strings and objects such as write, put etc.

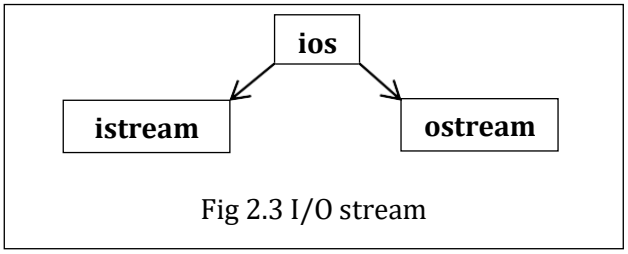


Fig 2.3 I/O stream

8. Standard Template Library:

The Standard Template Library (STL) is a set of template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. Working knowledge of template classes is a prerequisite for working with STL.

Four components of STL:

a. **Algorithms** b. **Containers** c. **Functions** d. **Iterators**

a. **Algorithms**:

The header algorithm defines a collection of functions specially designed to be used on a range of elements. They act on containers and provide means for various operations for the contents of the containers. e.g. searching, sorting, union, merge, heap, etc.

b. Containers: Fig.8

Containers or container classes store objects and data. There are in total seven standards “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors. e.g. vector (build in dynamic in nature), list (single & double), set, multiset, map, etc.

8.1 Vector:

Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

- begin() – Returns an iterator pointing to the first element in the vector
- end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
- size() – Returns the number of elements in the vector.
- max_size() – Returns the maximum number of elements that the vector can hold.
- assign() – It assigns new value to the vector elements by replacing old ones
- push_back() – It push the elements into a vector from the back
- pop_back() – It is used to pop or remove elements from a vector from the back.
- insert() – It inserts new elements before the element at the specified position
- erase() – It is used to remove elements from a container from the specified position or range.

8.2 Map:

Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key values.

- begin() – Returns an iterator to the first element in the map.
- end() – Returns an iterator to the theoretical element that follows the last element in the map.
- size() – Returns the number of elements in the map.

- max_size() – Returns the maximum number of elements that the map can hold.
- empty() – Returns whether the map is empty.
- pair insert(keyvalue, mapvalue) – Adds a new element to the map.
- erase(iterator position) – Removes the element at the position pointed by the iterator.
- erase(const g)– Removes the key-value ‘g’ from the map.
- clear() – Removes all the elements from the map.

CHAPTER – III

1. Asymptotic notation and Stack:

1.1 Asymptotic Notation:

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. There are mainly three asymptotic notations:

1.1.1 Big-O Notation (O-notation):

Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm (Fig. 9).

1.1.2 Omega Notation (Ω-notation):

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm (Fig. 10).

1.1.3 Theta Notation (Θ-notation):

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm (Fig. 3.1).

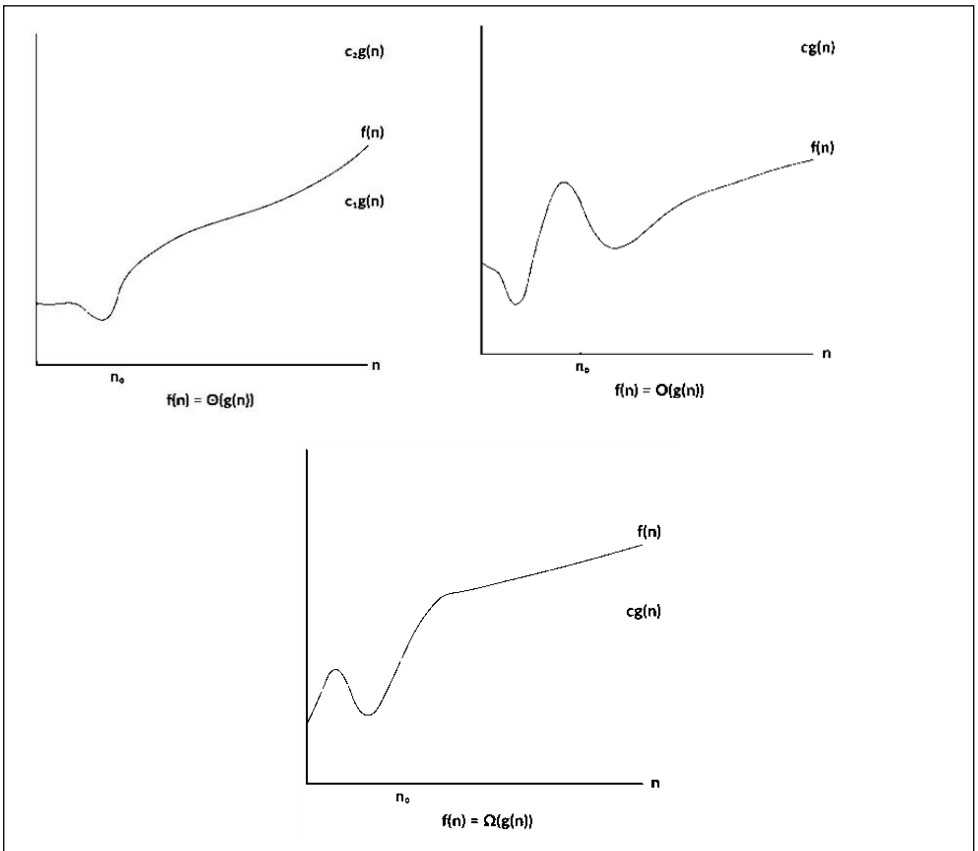


Fig 3.1 Asymptotic notations

1.2 Application of array:

a. Stack b. Queue c. Circular Queue d. Matrix

1.2.1 Stack:

It is a linear data structure that follows a particular order in which the operations are performed. This strategy states that the element that is inserted last will come out first. You can take a pile of plates kept on top of each other as a real-life example. The plate which we put last is on the top and since we remove the plate that is at the top, we can say that the plate that was put last comes out first.

Stack can be accessed by three methods:

a. Using Array b. Using Pointer c. Using STL

2. Infix, Postfix and Prefix & Queue:

2.1 Infix:

The typical mathematical form of expression that we encounter generally is known as infix notation. In infix form, an operator is written in between two operands.

For example: An expression in the form of A * (B + C) / D is in infix form. This expression can be simply decoded as: “Add B and C, then multiply the result by A, and then divide it by D for the final answer.”

2.2 Prefix:

In prefix expression, an operator is written before its operands. This notation is also known as “Polish notation”. For example, The above expression can be written in the prefix form as / * A + B C D. This type of expression cannot be simply decoded as infix expressions.

2.3 Postfix:

In postfix expression, an operator is written after its operands. This notation is also known as “Reverse Polish notation”. For example, The above expression can be written in the postfix form as A B C + * D /. This type of expression cannot be simply decoded as infix expressions.

2.4 Queue:

A queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.

We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.

Queue can be accessed by three methods:

a. Using Array b. Using Pointer c. Using STL

2.4.1 Circular Queue:

A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.

Circular Queue can be accessed by three methods:

a. Using Array b. Using Pointer

2.4.2 Priority Queue:

Priority queues are a type of container adapters, specifically designed such that the first element of the queue is either the greatest or the smallest of all elements in the queue and elements are in non-increasing order (hence we can see that each element of the queue has a priority (fixed order). However by default, the top element is always the greatest element. We can also change it to the smallest element at the top. Priority queues are built on the top to the max heap and uses array or vector as an internal structure.

3. List:

Lists are sequence containers that allow non-contiguous memory allocation. As compared to vector, the list has slow traversal, but once a position has been found, insertion and deletion are quick. List can be accessed by three methods:

a. Using Array b. Using Pointer c. Using STL

3.1 Linear Linked list:

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

3.2 Doubly Linked List:

A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.

3.3 Circular Linked List:

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which form a circle.

3.3.1 Circular singly linked list:

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

3.3.2 Circular Doubly linked list:

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

3.4 Application of linked list:

- Implementation of stacks and queues
- Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.
- Dynamic memory allocation: We use a linked list of free blocks.
- Maintaining a directory of names
- Performing arithmetic operations on long integers
- Manipulation of polynomials by storing constants in the node of the linked list

4. Trees and its application:

4.1 Trees:

A tree is a non-linear data structure that stimulates a tree structure with a root value and subset of children with parent nodes represented as a set of linked nodes.

4.2 Types of trees:

a. General tree b. Binary tree c. Binary search tree
d.AVL tree
e. Expression tree f. B tree g. B++ tree

4.3 Application of Trees:

- One reason to use trees might be because you want to store information that naturally forms a hierarchy.
- If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(log n) for search.
- trees guarantee an upper bound of O(log n) for insertion/deletion.
- Like Linked Lists and unlike Arrays, Pointer implementation of trees doesn't have an upper limit on number of nodes as nodes are linked using pointers.
- Store hierarchical data, like folder structure, organization structure, XML / HTML data.
- Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
- Heap is a tree data structure which is implemented using arrays and used to implement priority queues.
- B-Tree and B+ Tree: They are used to implement indexing in databases.
- Syntax Tree: Scanning, parsing, generation of code and evaluation of arithmetic expressions in Compiler design.
- K-D Tree: A space partitioning tree used to organize points in K dimensional space.

- Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks.

5. Binary Tree:

A binary tree is a tree data structure in which each parent node can have at most two children.

5.1 Types of binary tree:

5.1.1 Full binary tree:

A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree.

5.1.2 Perfect binary tree:

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

5.1.3 Complete binary tree:

A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

5.1.4 Balanced binary tree:

A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right sub-tree of any node differs by not more than 1.

5.2 Tree Traversal:

5.2.1 Preorder Traversal:

In the preorder traversal, the root node is processed first, followed by left subtree and then right subtree. The root gets processed before subtrees.

5.2.2 Inorder Traversal:

The inorder traversal processes the left subtree first, the root and finally right subtree.

5.2.3 Postorder Traversal:

In postorder the root is processed after the left and right subtree have been processed.

5.3 Binary Search Tree:

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

6. Graph:

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices and a set of edges.

6.1 Types of graph:

6.1.1 Null Graph

A graph is known as a null graph if there are no edges in the graph.

6.1.2 Trivial Graph

Graph having only a single vertex, it is also the smallest graph possible.

6.1.3 Undirected Graph

A graph in which edges do not have any direction. That is the nodes are unordered pairs in the definition of every edge.

6.1.4 Directed Graph

A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.

6.1.5 Weighted Graph

A graph in which the edges are already specified with suitable weight is known as a weighted graph. Weighted graphs can be further classified as directed weighted graphs and undirected weighted graphs.

6.2 Adjacency List:

This graph is represented as a collection of linked lists. There is an array of pointer which points to the edges connected to that vertex.

6.3 BFS and DFS:

6.3.1 Breadth First Search:

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

6.3.2 Depth First Traversal:

Depth First Traversal for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

7. Spanning tree and Hashing:

7.1 Spanning Tree:

A Spanning tree is a subset to a connected graph G, where all the edges are connected, i.e., one can traverse to any edge from a particular edge with or without intermediates. Also, a spanning tree must not have any cycle in it. Thus we can say that if there are N vertices in a connected graph then the no. of edges that a spanning tree may have is N-1.

7.1.1 Minimum Spanning Tree:

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to

the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

a. Prim's Algorithm:

It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

b. Kruskals Algorithm:

Given an undirected unweighted connected graph consisting of 'n' vertices and 'm' edges. The task is to find any spanning tree of this graph such that the maximum degree over all vertices is maximum possible. The order in which you print the output edges does not matter and an edge can be printed in reverse also i.e. (u, v) can also be printed as (v, u).

7.2 Hashing:

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

7.2.1 Types of Hashing:

7.2.1.1 Closed Hashing:

Open addressing. The types of closed hashing are:

a. Linear probing:

In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

The function used for rehashing is as follows: rehash(key) = (n+1)%table-size.

b. Quadratic probing:

If you observe carefully, then you will understand that the interval between probes will increase proportionally to the hash value. Quadratic probing is a method with the help of which we can solve the problem of clustering that was discussed above. This method is also known as the mid-square method. In this method, we look for the $(i^2)^{th}$ slot in the i^{th} iteration. We always start from the original hash location. If only the location is occupied then we check the other slots.

If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S

If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S and soon till we get empty location.

c. Double probing:

The intervals that lie between probes are computed by another hash function. Double hashing is a technique that reduces clustering in an optimized way. In this technique, the increments for the probing sequence are computed by using another hash function. We use another hash function hash2(x) and look for the i*hash2(x) slot in the i^{th} rotation.

If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S

If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S

and so on till we get empty location.

7.2.1.2 Open Hashing:

Close addressing.

8. Sorting and Heap:

8.1 Sorting:

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

8.1.1 Bubble Sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Follow the below steps for bobble sort:

1. Run a nested for loop to traverse the input array using two variables i and j, such that $0 \leq i < n-1$ and $0 \leq j < n-i-1$.
2. If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on.
3. Print the sorted array.

8.1.2 Insertion Sort:

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

To sort an array of size N in ascending order:

1. Iterate from arr[1] to arr[N] over the array.
2. Compare the current element (key) to its predecessor.
3. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

8.1.3 Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

1. Follow the below steps for selection sort:
2. Initialize minimum value(min_idx) to location 0.
3. Traverse the array to find the minimum element in the array.
4. While traversing if any element smaller than min_idx is found then swap both the values.
5. Then, increment min_idx to point to the next element.

6. Repeat until the array is sorted.

8.2 Heap:

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

8.2.1 Heap Sort:

First convert the array into heap data structure using heapify, than one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heapis greater than 1.

Follow the given steps for heap sort:

1. Build a max heap from the input data.
2. At this point, the maximum element is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of the heap by 1. Finally, heapify the root of the tree.
3. Repeat step 2 while the size of the heap is greater than 1.

8.2.2 Merge Sort:

Merge sort is one of the most efficient sorting algorithms. It is based on the divide-and-conquer strategy. Merge sort continuously cuts down a list into multiple sublists until each has only one item, then merges those sublists into a sorted list.

8.2.3 Quick Sort:

Quicksort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. For this reason, it is sometimes called partition-exchange sort.

Follow the given steps for Quick sort:

1. An array is divided into subarrays by selecting a pivot element (element selected from the array). While dividing the array, the pivot element should

be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
3. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

PROJECT

Project Title:

HOTEL MANAGEMENT SYSTEM

Description:

- Hotel management is the simple project for managing Hotel services.
- The project aims at developing a hotel management system using the CPP language that enables an organization to maintain its library.
- Hotel management is the process of organizing and managing details about customers,rooms and its services.
- Contact management system is purely for storing information about the customer.
- It saves a lot of time in arranging the data

Features:

- Room Details Function
- Check-in and check-out function
- Sign-up of managers
- Customer details
- Room details

Functions:

- Add customer
- Search Customer
- Rooms Booked
- Display all customers
- Delete customers
- Exit

Project Implementation:

Source Code:

```
#include<bits/stdc++.h>
#include<windows.h>
#include<fstream>
#include<iostream>
using namespace std;
bool  Empty=true;
//arrays
string costomerName;
int noofdays;
string *name;
int n;
string email;
char adhar[12];
string phone;
string dob;
string  age;
string temp;
int bottldrunkedbycustomer=0;
int billofcustomer=0;
int roomno;

int pepsi=0;
int fanta=0;
int sprite=0;
int fruty=0;
int lassi=0;
```

```
int beer=0;
int thumsup=0;
ofstream drinks;

class bill
{
   public:
   int a,b,c;
   void bills(){
            cout<<"Bill section\n";
            cout<<"Enter custumers NAME id\n";
            string id;
            cin>>id;
            ifstream customerdetail;
            customerdetail.open("cust_detail.txt");
            customerdetail>>costomerName;
            if(id==costomerName){
                     cout<<"Custumer Found\n";

            }
            else
            cout<<"Custumer does not found";
   }


};
void drink()
{
   int c;
```

```
cout<<"Choose the following options\n";
cout<<"1.Update the data of Drinks\n";
cout<<"2.See the remaining stock\n";
cin>>c;
if(c==1)
{
        drinks.open("drink.txt");
        cout<<"Enter the data in digits\n";
        cout<<"pepsi: ";
    cin>>pepsi;
        drinks<<pepsi<<endl;
        cout<<"Fanta: ";
        cin>>fanta;
        drinks<<fanta<<endl;
        cout<<"Sprite: ";
        cin>>sprite;
        drinks<<sprite<<endl;
        cout<<"Fruti: ";
        cin>>fruty;
        drinks<<fruty<<endl;
        cout<<"Lassi: ";
        cin>>lassi;
        drinks<<lassi<<endl;
        cout<<"Beer: ";
        cin>>beer;
        drinks<<beer<<endl;
        cout<<"Thums up: ";
        cin>>thumsup;
```

```
        drinks<<thumsup<<endl;
    }
    drinks.close();
    if(c==2){
        ifstream drinks;
        drinks.open("drink.txt");
        cout<<"COLD DRINKS remaining in stock are: \n";
        cout<<"pepsi: ";
        drinks>>pepsi;
        cout<<pepsi<<endl;
        cout<<"Fanta: ";
        drinks>>fanta;
        cout<<fanta<<endl;
        cout<<"Sprite: ";
        drinks>>sprite;
        cout<<sprite<<endl;
        cout<<"Fruti: ";
        drinks>>fruty;
        cout<<fruty<<endl;
        cout<<"Lassi: ";
        drinks>>lassi;
        cout<<"Beer: ";
        drinks>>beer;
        cout<<beer<<endl;
        cout<<"Thums up: ";
        drinks>>thumsup;
        cout<<thumsup<<endl;
    }
    drinks.close();
```

```
}
ofstream customerdetail;
ofstream floor1;
ofstream floor2;
ofstream floor3;
ofstream roomrent;
ofstream drinktype;
ofstream hotdrink;
void floor_1();
void floor_2();
void floor_3();



void emptyroomInfo()
{
   cout<<"Current room Status\n";
   cout<<"Total Floors in a hotel:3\n";
   cout<<"Total rooms are:10 on each floor\n";
   // cout<<"Rooms Occupied on first floor are :";
   int choice;
   cout<<"Choose the options to view information about each floor\n";
   cout<<"\t\n1.first floor\t\n2.Second floor\t\n3.Third floor";
   cin>>choice;
   switch(choice){
         case 1:
         floor_1();
         break;
         case 2:
```

```
            floor_2();
            break;
            case 3:
            floor_3();
            break;
            case 4:
            cout<<"Invalid choice\n";
    }

}

void  floor_1()

    {
            ifstream floor1;
            floor1.open("floor1.txt");
            cout<<"Rooms Booked on floor 1 are : "<<endl;
            while(!floor1.eof())
            {
                    // cin.ignore();
                    floor1>>roomno;
                    cout<<roomno<<endl;
            }
            floor1.close();
            cout<<endl;
    }
    void  floor_2()
    {
            ifstream floor2;
```

```
        floor2.open("floor2.txt");
        cout<<"Rooms Booked on floor 2 are : "<<endl;
        while(!floor2.eof())
        {
            // cin.ignore();
            floor2>>roomno;
            cout<<roomno<<endl;
        }
        floor2.close();
        cout<<endl;
    }
    void  floor_3()
    {
        ifstream floor3;
        floor3.open("floor3.txt");
        cout<<"Rooms Booked on floor 3 are : "<<endl;
        while(!floor3.eof())
        {
            // cin.ignore();
            floor3>>roomno;
            cout<<roomno<<endl;
        }
        floor3.close();
        cout<<endl;
    }

class order
{
    public:
```

```
int cost;
void drinks()
{
        int type;
        cout<<"Place a order for drink\n";
        cout<<"Choose the type of DRINK!!!";
        cout<<"\t\n1. \nCOLD 2.HOT";
        cout<<endl;

        cin>>type;
        switch(type)
        {
                case 1:
                cold();
                break;
                case 2:
                hot();
                break;
                case 3:
                cout<<"Invalid choice!!!\n";
        }
}
 void cold()
{
        ka:
        cout<<"Choose a drink\n";
        cout<<"1.FRESH GINGER TEA\n ";
        cout<<"2.FRUIT TEA\n ";
        cout<<"3.FRESH MINT TEA\n ";
```

```
cout<<"4.HOT CHOCOLATE\n";
cout<<"5.COFFE\n";
cout<<"6.HOT LEMON\n";
cout<<"7.GREEN TEA\n";
cout<<"8.CHAI\n";
system("cls");
int typ;
cin>>typ;
if(typ==1)
{
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Ginger tea selected\n";
        drinktype<<"Ginger tea"<<endl;
        cout<<endl;
        drinktype.close();
}
else if(typ==2){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Fruit tea selected\n";
        drinktype<<"Fruit tea"<<endl;
        cout<<endl;
        drinktype.close();
}
else if(typ==3){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Fresh mint tea selected\n";
        drinktype<<"Fresh mint tea"<<endl;
        drinktype.close();
}
```

```
else if(typ==4){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Hot chocolate selected\n";
        drinktype<<"Hot chocolate"<<endl;
        drinktype.close();
}
else if(typ==5){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Coffe selected\n";
        drinktype<<"Coffe"<<endl;
        drinktype.close();
}
else if(typ==6){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Hot lemon Selected\n";
        drinktype<<"Hot Lemon"<<endl;
        drinktype.close();
}
else if(typ==7){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Green tea selected\n";
        drinktype<<"Green Tea"<<endl;
        drinktype.close();
}
else if(typ==8){
        drinktype.open("drinktype.txt",ios::app);
        cout<<"Chai Selected\n";
        drinktype<<"Chai"<<endl;
        drinktype.close();
```

```
        }
        else if(typ<1 && typ>8){
                cout<<"Invalid Choice!!!\n";
                cout<<"Choose proper drinkn\n";
                goto ka;
        }

}

void  hot()
{
        la:
        cout<<"choose form the following menu\n";
        cout<<"1.honey citrus mint tea\n";
        cout<<"2.Streamed apple juice\n";
        cout<<"3.Flat white\n";
        cout<<"4.Cinnamon Dolce latte\n";
        cout<<"5.Peppermint Mocha\n";
        cout<<"6.Matcha tea latte\n";
        cout<<"7.Caffe Americano\n";
        cout<<"8.Pumpkin Spice Latte\n";
        int typ;
        cin>>typ;
        hotdrink.open("hotdrink.txt",ios::app);
        if(typ==1)

        {
                cout<<"honey citrus mint tea SELECTED\n";
                hotdrink<<"honey citrus mint tea"<<endl;
```

```
        hotdrink.close();
}
else if(typ==2)

{
        cout<<"Streamed apple juice SELECTED\n";
        hotdrink<<"Streamed apple juice"<<endl;
        hotdrink.close();
}
else if(typ==3)

{
        cout<<"Flat white SELECTED\n";
        hotdrink<<"Flat white"<<endl;
        hotdrink.close();
}
else if(typ==4)

{
        cout<<"Cinnamon Dolce latte SELECTED\n";
        hotdrink<<"Cinnamon Dolce latte"<<endl;
        hotdrink.close();
}
else if(typ==5)

{
        cout<<"Peppermint Mocha SELECTED\n";
        hotdrink<<"Peppermint Mocha"<<endl;
        hotdrink.close();
```

```
        }
        else if(typ==6)

        {
                cout<<"Matcha tea latte SELECTED\n";
                hotdrink<<"Matcha tea latte"<<endl;
                hotdrink.close();
        }
        else if(typ==7)
        {
                cout<<"Caffe Americano SELECTED\n";
                hotdrink<<"Caffe Americano"<<endl;
                hotdrink.close();
        }
        else if(typ==8)
        {
                cout<<"Pumpkin Spice Latte SELECTED\n";
                hotdrink<<"Pumpkin Spice Latte"<<endl;
                hotdrink.close();
        }
        else if(typ<1 && typ>8){
                cout<<"Invalid Choice!!!\n";
                cout<<"Choose proper drinkn\n";
                goto la;
        }


}
```

```
};
void print()
{
   cout<<"Information of customer "<<endl;
   cout<<endl;
   cout<<endl;
   cout<<"Name of customer is:";
   cout<<costomerName;
   cout<<"\n";
   cout<<"email of customer is:";
   cout<<email;
   cout<<endl;
   cout<<"Aadhar number of customer:";
   cout<<adhar<<endl;
    cout<<"Mobile number of customer is: ";
   cout<<phone<<endl;
   cout<<"DOB of customer is: ";
   cout<<dob<<endl;
   cout<<"Age of customer is: ";
   cout<<age;
   cout<<endl;
   cout<<endl;
   temp=costomerName;

}
void infocustomer()
{
   ifstream customerdetail;
   customerdetail.open("cust_detail.txt");
```

```
    int s=0;


   cout<<"--------------------------------------------------------------------------------
"<<endl;
   cout<<"                    WELCOME TO RED ROCK INN
"<<endl;
   cout<<"--------------------------------------------------------------------------------
"<<endl;
   cout<<"The information of customers is getting ready...\n";
   cout<<"Please wait for some time :)\n";


   while(!customerdetail.eof()){
           // int i=1;



    customerdetail>>costomerName;
   customerdetail>>email;
   customerdetail>>adhar;
   customerdetail>>phone;
   customerdetail>>dob;
   customerdetail>>age;
   if(temp==costomerName)
   {
           break;
   }


   print();
   }
   if(costomerName.empty())
   {
```

```
        cout<<"list is empty"<<endl;
    }
    customerdetail.close();
}
int search(string Name)
 {
    ifstream  customerdetail;
    customerdetail.open("cust_detail.txt");
    while (!customerdetail.eof()) {

    customerdetail>>costomerName;
    customerdetail>>email;
    customerdetail>>adhar;
    customerdetail>>phone;
    customerdetail>>dob;
    customerdetail>>age;
        if (Name==costomerName ) {

            print();
                    temp=costomerName;
                    customerdetail.close();
            return 1;
        }

    }
            system("cls");
    cout<<"Sorry custumer Not Availiable!!!"<<endl;
    customerdetail.close();
    return -1;
```

```
}
void deletes(string Name)
{
   int num = search( Name);
   if (num==1) {

      ofstream write;
         write.open("temps.txt");
      ifstream customerdetail;
      customerdetail.open("cust_detail.txt");
      while (!customerdetail.eof()) {
                  customerdetail>>costomerName;
       customerdetail>>email;
         customerdetail>>adhar;
        customerdetail>>phone;
        customerdetail>>dob;
         customerdetail>>age;

         if ( costomerName!=Name)
                  {

            write<<"\n"<<costomerName;
       write<<"\n"<<email;
         write<<"\n"<<adhar;
        write<<"\n"<<phone;
        write<<"\n"<<dob;
         write<<"\n"<<age;
         }
      }
```

```
        write.close();
        customerdetail.close();
        remove("cust_detail.txt");
        rename("temps.txt", "cust_detail.txt");

        cout << "\n\tData deleted successfuly"<<endl;
    }
    else {
                system("cls");
        cout << "\n\tRecord not deleted Something do Wrong";
    }
}
void newcustomer()
{

   int i;
   la:
   cout<<"Enter no of customer\n";
   cin>>n;
   if(n<4){
          customerdetail.open("cust_detail.txt",ios::app);

          for(i=0;i<n;i++){

                cout<<"Name of"<<(i+1)<<"customer:";
                cin>>costomerName;
                customerdetail<<costomerName<<endl;
                cout<<"Enter the email of "<<i+1<<" customer:";
                cin>>email;
```

```
        customerdetail<<email<<endl;
        cout<<"Enter the aadhar number of "<<i+1<<" customer:";
        cin>>adhar;
        customerdetail<<adhar<<endl;
        cout<<"Enter the mobile number of "<<i+1<<" customer:";
        cin>>phone;
        customerdetail<<phone<<endl;
        cout<<"Enter the date of birth of "<<i+1<<" customer:";
        cin>>dob;
        customerdetail<<dob<<endl;
        cout<<"Enter the age of "<<i+1<<" customer:";
        cin>>age;
        customerdetail<<age<<endl;

        cout<<endl;
        cout<<endl;

}
customerdetail.close();
cout<<"Select floor and room number to live:\n";
cout<<"1.for floor 1\n";
cout<<"2.for floor 2\n";
cout<<"3.for floor 3\n";
        int floor;
        cin>>floor;
        if(floor==1)
        {

                floor1.open("floor1.txt",ios::app);
```

```
                cout<<"Enter room no to live.:1 to 10: \n";
            cin>>roomno;
                floor1<<roomno<<endl;
                cout<<endl;
                floor1.close();
            }

            else if(floor==2)
            {
                floor2.open("floor2.txt",ios::app);
                cout<<"Enter room no.: from 11 to 20: ";
                cin>>roomno;
                floor2<<roomno<<endl;
                cout<<endl;
                floor2.close();
            }
            else if(floor==3){
                floor3.open("floor3.txt",ios::app);
                cout<<"Enter a room no. to live:from 21 to 30: \n";
                cin>>roomno;
                floor3<<roomno<<endl;
                cout<<endl;
                floor3.close();
            }

    }
    else
    {
        cout<<"Maximum 3 people can live in one room"<<endl;
```

```
            cout<<"Please book another room for remaining customers\n";
            goto la;
    }
    int type;
    cout<<"Choose the type of room:\n";

    cout<<"1.Simple room, rent: Rs1199/day\n";
    cout<<"2.Premium room, rent:Rs1690/day\n ";
    cout<<"3.Ultra Premium room, rent: Rs2490/day\n";
    cout<<"4.Luxury Room, rent: Rs4490/day\n";

    cin>>type;
    roomrent.open("rent.txt",ios::app);
    cout<<"Enter no.of days you want to live\n";
    cin>>noofdays;
    customerdetail<<noofdays<<endl;

    if(type==1)
    {
            cout<<"Total rent of simple room for "<<noofdays<<"is
Rs:"<<1199*(noofdays)<<endl;
            customerdetail<<(1199*(noofdays))<<endl;
       // customerdetail.close();
    }
    else if(type==2)
    {
            cout<<"Total rent of premium room for "<<noofdays<<"is
Rs:"<<1690*(noofdays)<<endl;
            customerdetail<<(1690*(noofdays))<<endl;
       // customerdetail.close();
```

```
    }
    else if(type==3)
    {
            cout<<"Total rent of ultra premium room for "<<noofdays<<"is
Rs:"<<2490*(noofdays)<<endl;
            customerdetail<<(2490*(noofdays))<<endl;
       // customerdetail.close();
    }
    else if(type==4)
    {
            // customerdetail.open("cust_detail.txt",ios::app);
            cout<<"Total rent of Luxury room for "<<noofdays<<"is
Rs:"<<4490*(noofdays)<<endl;
            customerdetail<<(4490*(noofdays))<<endl;
    //  customerdetail.close();
    }
    roomrent.close();
    cout<<"Room booked Sucessfully:)\n\n";

}
int login()
{
    int count;
    string userid,password,id,pass;
// system("cls");
    cout<<"\t\tPlease enter your username and password-"<<endl;
    cout<<"\t\tUSERNAME-";
    cin>>userid;
    cout<<"\t\tPASSWORD-";
    cin>>password;
```

```
   ifstream input("record.txt");
   while(input>>id>>pass)
   {
        if(id==userid&&pass==password)
        {
             count=1;
        //   system("cls");
        }
   }
   input.close();
   if(count==1)
   {    system("cls");
        cout<<"\t\tYou login is successfully"<<endl;

        cout<<"########################WELCOME-
"<<userid<<"############################## "<<endl;
             return 1;
   }
   else
   {
             system("cls");
             cout<<"\t\tLogin error please check your password and
username"<<endl;
             return 0;
   }

}
void registration()
{
        string ruserid,rpassword,rid,rpass;
```

```
            //system("cls");
            cout<<"\t\t\tEnter a username-";
            cin>>ruserid;
            cout<<"\t\t\tEnter a password-";
            cin>>rpassword;

            ofstream f1("record.txt",ios::app);
            f1<<ruserid<<' '<<rpassword<<endl;
                  system("cls");
            cout<<"\t\t\tRegistration is successful!!!\n"<<endl;
   }
void forgot()
{
   int option;
// system("cls");
            do
            {

    cout<<"press 1 to Search your password by username"<<endl;
    cout<<"press 2 for Go Back"<<endl;
   cout<<"\t\t\tEnter your choice-";
   cin>>option;



   switch(option)
   {
            case 1:
            {
```

```
                    system("cls");
            int count=0;
            string suserid,sid,spass;
            cout<<"\t\t\tEnter the username which you rememberd:";
            cin>>suserid;
            ifstream f2("record.txt");
            while(f2>>sid>>spass)
            {
                if(sid==suserid)
                    count=1;
            }
            if(count==1)
            {
                    system("cls");

                cout<<"\n\nYour account is found!!!!\n";
                cout<<"\n\nYour password is-"<<spass<<endl;
            }
            else
            {
                    system("cls");
            cout<<"\t\tSorry your account is not found!!\n"<<endl;
      }
      }
      case 2:
            break;
      default:
            cout<<"Please Enter correct choice"<<endl;
}
```

```
}while(option!=2);
}
int main()
{
   int ch,choice;
    string temp;
   do
   {
       cout<<"--------------------------------------------------------------------------------
--"<<endl;
   cout<<"                    WELCOME TO RED ROCK INN
"<<endl;
   cout<<"--------------------------------------------------------------------------------
"<<endl;


   cout<<"_____________________________________________________________________
_____"<<endl;
      cout<<"\t\t\tWELCOME TO LOGIN PAGE"<<endl;
   cout<<"_____________________________________________________________________
_____"<<endl;
   cout<<"\t1.Manager Login"<<endl;
   cout<<"\t2.Register"<<endl;
   cout<<"\t3.Forgot password"<<endl;
   cout<<"\t4.Close Application"<<endl;
   cout<<"\t\tEnter your choice-";
   cin>>choice;
   cout<<endl;
   switch(choice)
   {
    case 1:
```

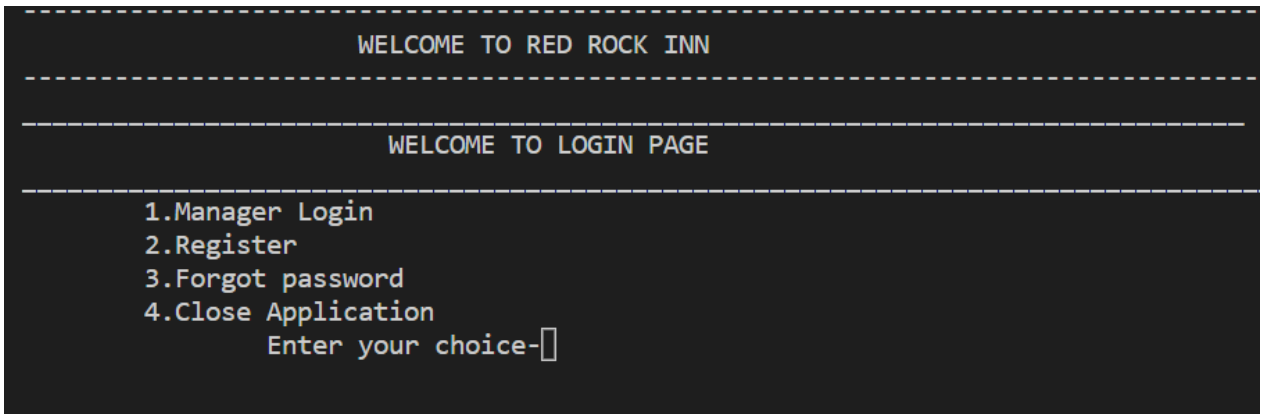
```
 if(login())
 {
do
{


 cout<<"--------------------------------------------------------------------------------
"<<endl;
 cout<<"                    WELCOME TO RED ROCK INN
"<<endl;
 cout<<"--------------------------------------------------------------------------------
"<<endl;


 cout<<"Choose the following Actions\n";
 cout<<"1.Get inforamtion about empty Rooms\n";
 // cout<<"2.To get information about Empty Rooms on each floor\n";
 cout<<"3.Get information about Remaining stock of drinks\n";
 cout<<"4.To add new customer\n";
 cout<<"5.Get information about customer in room\n";
 cout<<"6.Place a order for a custumer\n";
 cout<<"7.Search a custumer record\n";
 cout<<"8.Delete a customer record\n";
 cout<<"9.Exit from Application\n";
 cin>>ch;
 system("cls");
 switch(ch)
 {
        case 1:
        emptyroomInfo();
        break;
```

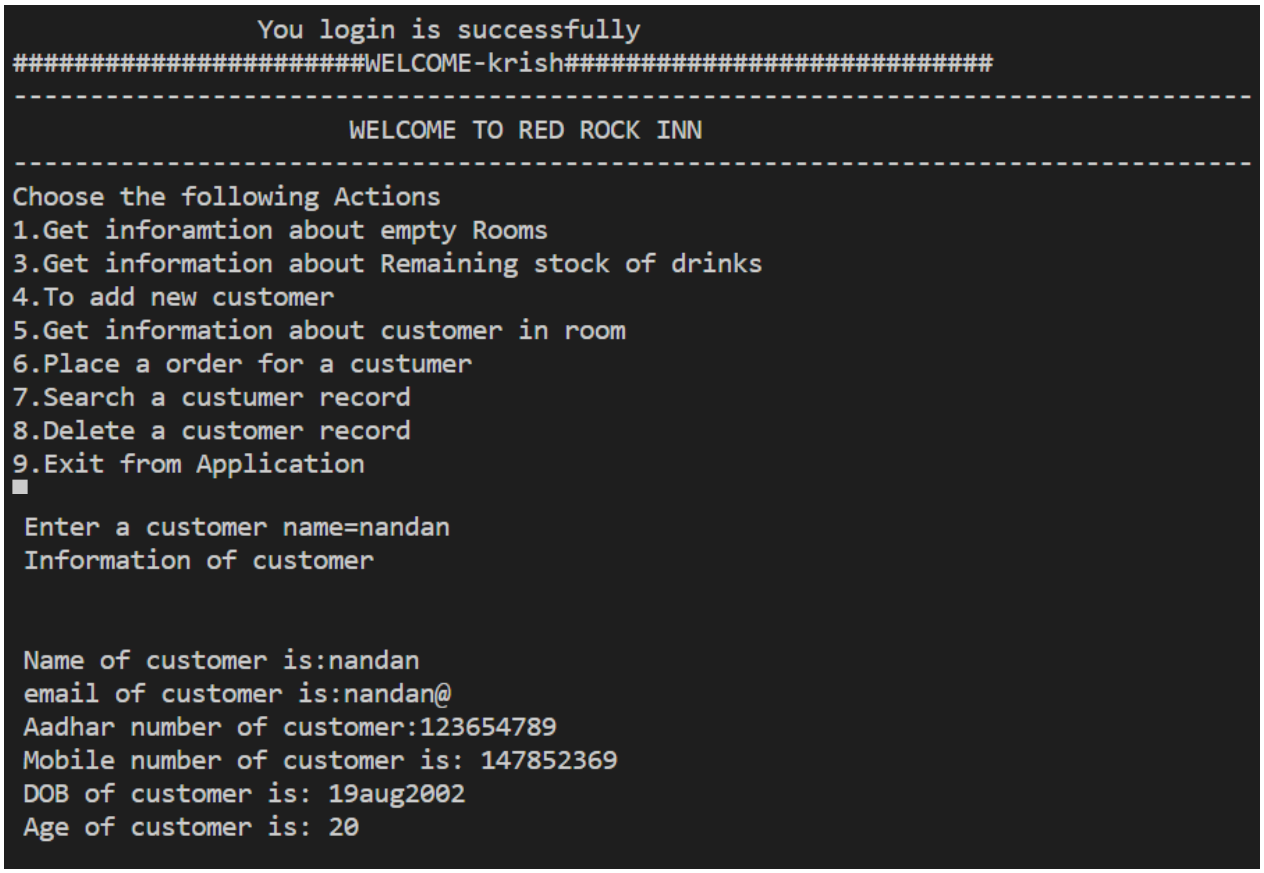
```
        // case 2:
        // emptyroomInfoeachfoor();
        // break;
        case 3:
    drink();
        break;
        case 4:
        newcustomer();
        break;
        case 5:
        infocustomer();
        break;
        case 6:
         order o;
        o.drinks();
        break;
    case 7:
    cout<<"Enter a customer name=";
    cin>>temp;
    search(temp);
    break;
         case 8:
    cout<<"Enter a customer name=";
    cin>>temp;
    deletes(temp);
    break;
        case 9:
        cout<<"Logout Successfully\n";
        break;
```

```
            default:
            cout<<"Please enter correct choice:)\n";
        }
        }while(ch!=9);
         }
         break;
            case 2:
                        system("cls");
                registration();
                break;
            case 3:
                        system("cls");
                forgot();
                break;
            case 4:
                        system("cls");
               cout<<"Thank You "<<endl;
               break;
            default:
                        system("cls");

                cout<<"Please enter correct choice"<<endl;
        }


    }while(choice!=4);
```

}

Output:


WELCOME TO RED ROCK INN
WELCOME TO LOGIN PAGE
1.Manager Login
2.Register
3.Forgot password
4.Close Application
Enter your choice-


You login is successfully
##########WELCOME-krish##########
WELCOME TO RED ROCK INN
Choose the following Actions
1.Get inforamtion about empty Rooms
3.Get information about Remaining stock of drinks
4.To add new customer
5.Get information about customer in room
6.Place a order for a custumer
7.Search a custumer record
8.Delete a customer record
9.Exit from Application
Enter a customer name=nandan
Information of customer
Name of customer is:nandan
email of customer is:nandan@
Aadhar number of customer:123654789
Mobile number of customer is: 147852369
DOB of customer is: 19aug2002
Age of customer is: 20

```
Current room Status
Total Floors in a hotel:3
Total rooms are:10 on each floor
Choose the options to view information about each floor

1.first floor
2.Second floor
3.Third floor1
Rooms Booked on floor 1 are :
5
2
```

CONCLUSION

From my Industrial Training at ‘IFS India Mercantile Pvt. Ltd. Pune’, I was able to get a better understanding of how the software industry works and how effective it is. I enjoyed working on the data structure and oops projects. However, I still have a long way to go in understanding the deeper concepts of the data science. From working on the technical aspects of a data science project to presenting my work in a way that everyone can understand, this has been a great experience!

Overall, I found the Data Structure and Object-Oriented Programming internship experience to be positive, and I'm sure I will be able to use skills I learnt in my career later.

Reference

[1] https://www.geeksforgeeks.org/c-plus-plus/?ref=shm

[2] https://www.javatpoint.com/cpp-tutorial

[3] https://www.programiz.com/dsa/quick-sort

[4] https://www.codingninjas.com/codestudio/library/infix-postfix-and-prefix-conversion

[5] Some You tube Channels Like Code with harry ,love Babber ,Jenny's Lecture And Tutorials