

INTRODUCTION TO DSA WITH C++

FIFTH FORCE

Submitted in partial fulfilment of the requirements for the award of degree of

B .Tech (CSE)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



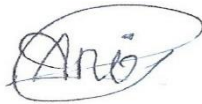
From 05/26/2022 to 07/10/2022

SUBMITTED BY

Name of student: Nirbhay Singh

Registration Number: 12019256

Signature of the student:



To whom so ever it may concern

I, **Nirbhay Singh, 12019256**, hereby declare that the work done by me on “**Introduction to DSA with C++**” from **May, 2022** to **July, 2022**, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **B .Tech (CSE)**.

Name of the Student: Nirbhay Singh

Signature of the student

A handwritten signature in blue ink, appearing to read 'Nirbhay', enclosed within a hand-drawn oval border.



ACKNOWLEDGMENT

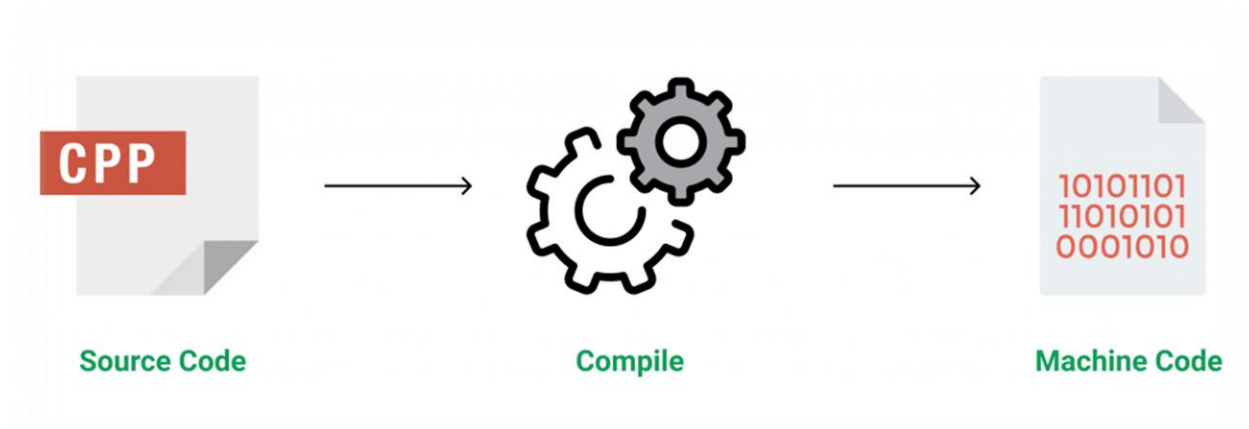
I have taken efforts in this Project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I thanks to FIFTH FORCE for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the Project. My thanks and appreciation also go to my college (Lovely Professional University) who gave me this opportunity to do this project and gain knowledge and be able to complete the objective of this project.

INTRODUCTION

About C++:

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language.



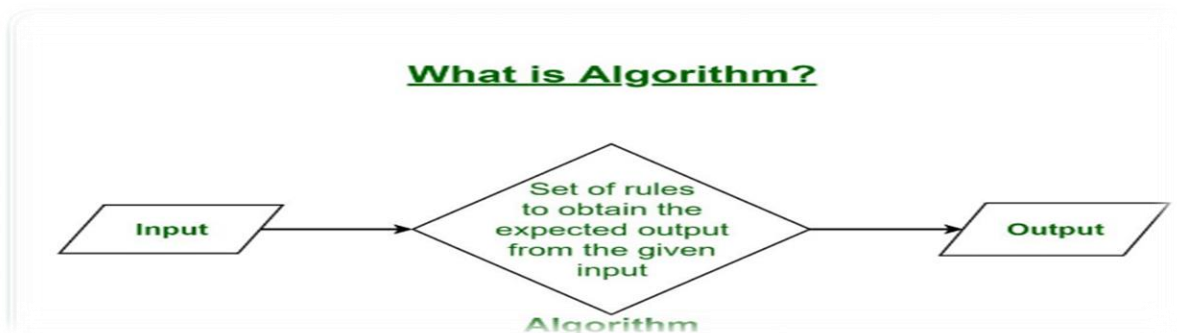
C++ is a middle-level language rendering it the advantage of programming low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

About Data Structure:

Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's **name** "Virat" and **age** 26. Here "Virat" is of **String** data type and 26 is of **integer** data type.

About Algorithm:

The word Algorithm means "A set of rules to be followed in calculations or other problem-solving operations" Or "A procedure for solving a mathematical problem in a finite number of steps that frequently involves recursive operations".



C++ Programming Language

C++, as we all know is an extension to C language and was developed by Bjarne Stroustrup at bell labs.

C++ is a general-purpose programming language and widely used nowadays for competitive programming. It has imperative, object-oriented and generic programming features. C++ runs on lots of platform like Windows, Linux, Unix, Mac, etc.

Input/Output Streams

COUT<<

It is used to print anything on screen, same as printf in C language. cin and cout are same as scanf and printf, only difference is that you do not need to mention format specifiers like, %d for int etc, in cout & cin.

<< is called insertion operator.

Example:

int a=10;	
printf("a= %d",a);	(In C programming)
cout<<"a="<<a;	(In C++ programming)

CIN>>

It is a predefined object and represents the standard input stream and this input stream is used to read the data or take value from user.

>> is called extraction operator.

Example:

```
int a,b;
cin>>a>>b;
```

Comments

In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program. Comments are statements that are not executed by the compiler and interpreter.

In C/C++ there are two types of comments:

- Single line comment
- Multi-line comment

Comments

// Single line comment

/* Multi-line comment */

C++ Data Types

A data type is to let know the variable, what type of element it is and definitely going to determine the memory allocation of that variable. Each data type has a different memory allocation.

C++ supports the following data types:

- Primary or built in or Fundamental data type
- Derived data types
- User defined data types

Primitive Data Types: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc. Primitive data types available in C++ are:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

Derived Data Types: The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

- Function
- Array
- Pointer
- Reference

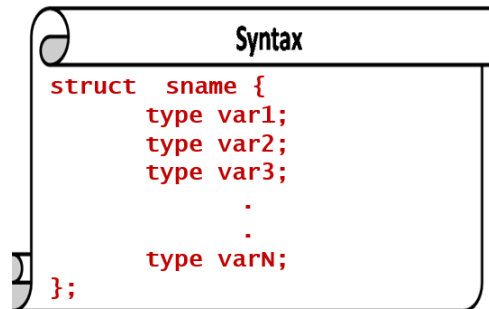
Abstract or User-Defined Data Types: These data types are defined by the user itself. Like, as defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

- Class
- Structure
- Union
- Enumeration

Structures

Structures are derived data types. It is a group of data items of different data types held together in a single unit.

- Collections of related variables under one name
- Can contain variables of different data types
- Commonly used to define records to be stored in files.



Syntax

```
struct sname {  
    type var1;  
    type var2;  
    type var3;  
    .  
    .  
    type varN;  
};
```

- struct is a keyword to define a structure.
- sname is the name given to the structure/structure tag.
- type is a built-in data type.
- var1,var2,var3,.....,varN are elements of structure being defined.
- ; semicolon at the end.

Union

Union is similar as structure. The major distinction between them is in terms of storage. In structure each member has its own storage location whereas all the members of union uses the same location. The union may contain many members of different data type but it can handle only one member at a time union can be declared using the keyword union

Union tagname

```
{  
    Datatype1 mem1;  
    Datatype2 mem2;  
    Datatype3 mem3;  
    .....  
    .....  
    DatatypeN memN;  
};
```

- union is a keyword to define a union
- tagname is name to union
- Datatype1, Datatype2 are data types
- mem1, mem2 are variables
- ; semicolon at end

Variable In C++

A variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location; all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

Rules For Declaring Variable:

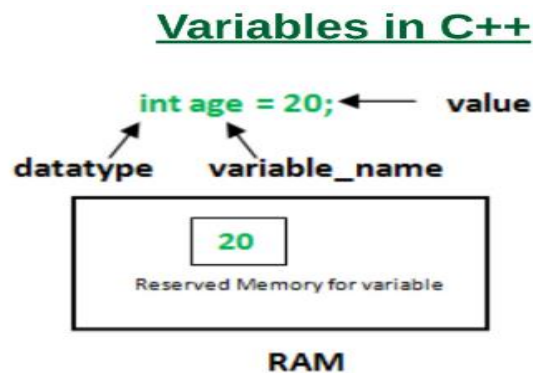
- 1.The name of the variable contain letters, digits, and underscores.
- 2.The name of the variable are case sensitive (ex Arr and arr both are different variable).
- 3.The name of the variable does not contain any whitespace and special characters (ex #,\$,%,* etc).
- 4.All the variable name must begin with a letter of the alphabet or an underscore (_).
- 5.We cannot used C++ keyword (ex float, double, class) as a variable name.

Declaration of Variable

A typical variable declaration is of the form:

- Declaring a single variable
type variable_name;
- Declaring multiple variables:
type variable1_name, variable2_name, variable3_name;

A variable name can consist of alphabets (both upper and lower case), numbers and the underscore '_' character. However, the name must not start with a number.



Decision Making in C++ (if, if. Else, Nested if, if-else-if)

Decision-making statements in programming languages decide the direction of the flow of program execution. Decision-making statements available in C or C++ are:

if statement

- if. Else statements
- if statements
- nested if statements
- switch statements
- Jump Statements:
 - break
 - continue
 - goto
 - return

if statement in C++

if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not

Syntax:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

if-else in C++

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.else execute the code if condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

nested-if in C++

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement.

Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition2)
    {
        // Executes when condition2 is true
    }
}
```

Jump Statements in C++

These statements are used in C or C++ for the unconditional flow of control throughout the functions in a program. They support four types of jump statements:

break: This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

Syntax:

```
break;
```

continue: This loop control statement is just like the break statement. The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

Syntax:

```
continue;
```

goto: The goto statement in C/C++ also referred to as unconditional jump statement can be used to jump from one point to another within a function.

Syntax:

```
goto label1;      label1;
..                ...
..                ..
..                ..
label1;           goto label1;
```

return:

The return in C or C++ returns the flow of the execution to the function from where it is called.

Syntax:

```
return[expression];
```

Loops in C++

In programming, sometimes there is a need to perform some operation more than once or (say) n number of times. Loops come into use when we need to repeatedly execute a block of statements.

There are mainly two types of loops:

- Entry Controlled loop
- Exit Controlled Loops

S.No. Loop Type and Description

1. while loop – First checks the condition, then executes the body.
2. for loop – firstly initializes, then, condition check, execute body, update.
3. do-while – firstly, execute the body then condition check

for Loop

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.

Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

while Loop

while loops are used in situations where we do not know the exact number of iterations of the loop beforehand. The loop execution is terminated on the basis of the test conditions.

Syntax:

```
initialization expression;
while (test_expression)
{
    // statements
    update_expression;
}
```

do-while loop

In do-while loops also the loop execution is terminated on the basis of test conditions. But here condition is tested at end.

Syntax:

```
initialization expression;
do
{
    // statements
    update_expression;
} while (test_expression);
```

Operators in C++

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands.

For example, '+' is an operator used for addition, as shown below:

```
c = a + b;
```

Here, '+' is the operator known as the addition operator and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'.

C++ has many built-in operators and can be classified into 6 types:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Other Operators

Unary operators in C++

Unary operators: are operators that act upon a single operand to produce a new value.

Types of unary operators:

- unary minus(-)
- increment(++)
 - prefix
 - postfix
- decrement(- -)
 - prefix decrement
 - postfix decrement
- NOT(!)
- Addressof operator(&)
- sizeof()

new and delete Operators in C++ For Dynamic Memory

new operator

The new operator denotes a request for memory allocation on the Free Store

Syntax:

```
pointer-variable = new data-type;
```

delete operator

It is used to deallocate dynamically allocated memory.

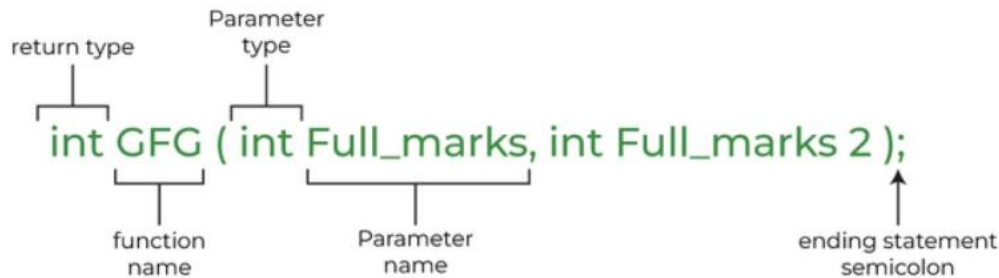
Syntax:

```
delete pointer-variable;
```

Functions in C++

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.

Syntax:



Types of Functions

User Defined Function

User Defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs. They are also commonly known as “tailor-made functions” which are built only to satisfy the condition in which the user is facing issues meanwhile reducing the complexity of the whole program.

Library Function

Library functions are also called “builtin Functions“. These functions are a part of a compiler package that is already defined and consists of a special function with special and different meanings. Builtin Function gives us an edge as we can directly use them without defining them whereas in the user-defined function we have to declare and define a function before using them. For Example: `sqrt()`, `setw()`, `strcat()`, etc.

Parameter Passing to Functions

- The parameters passed to function are called actual parameters.
- The parameters received by the function are called formal parameters

There are two most popular ways to pass parameters:

- **Pass by Value:** In this parameter passing method, values of actual parameters are copied to the function’s formal parameters and the two types of parameters are stored in different memory locations. So, any changes made inside functions are not reflected in the actual parameters of the caller.
- **Pass by Reference:** Both actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in the actual parameters of the caller.

Pointers in C++

A pointer is a variable that holds the memory address of another variable. A pointer needs to be dereferenced with the * operator to access the memory location it points to.

Syntax:

datatype *var_name;



- To access address of a variable to a pointer, we use the unary operator & (ampersand) that returns the address of that variable. For example, &x gives us address of variable x.
- To access the value stored in the address we use the unary operator (*) that returns the value of the variable located at the address specified by its operand. This is also called Dereferencing

Object Oriented Programming

In C++, rather than creating separate variables and functions, we can also wrap these related data and functions in a single place (by creating objects). This programming paradigm is known as object-oriented programming.

Class

- A class is a blueprint for the object.
- We can think of a class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Create a Class

- A class is defined in C++ using keyword `class` followed by the name of the class.
- The body of the class is defined inside the curly brackets and terminated by a “;” at the end.

Syntax:

```
class className
{
// some data
// some functions
};
```

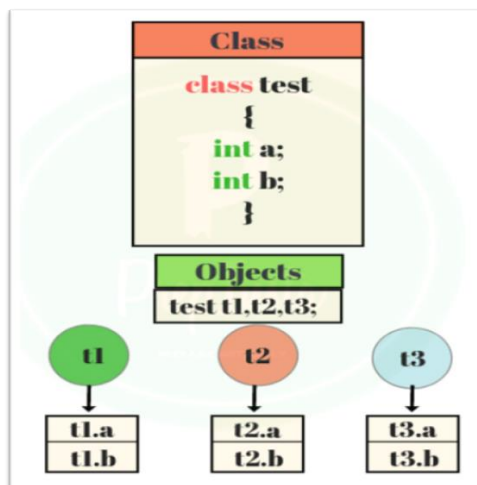
Object

An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e., an object is created) memory is allocated.

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.

Syntax:

ClassName ObjectName;



Accessing a Data Member

Accessing a data member depends solely on the access control of that data member. If its public, then the data member can be easily accessed using the direct member access (.) operator with the object of that class.

For Example:

```
class A
{
public:
int x;
Void func();
}obj1;
main()
{
A obj2;
obj1.x=10;
obj2.x=20;
obj1.func();
---
}
```

- Annotation for `public:`: Access Specifier : will discuss in the next slide
- Annotation for `}obj1;`: 1st Method to declare object
- Annotation for `A obj2;`: 2nd Method to declare object
- Annotation for `obj1.x=10;`, `obj2.x=20;`, and `obj1.func();`:
 - Obj_name.Var1
 - Obj_name.function()
 - Accessing data members using obj

Access Specifier

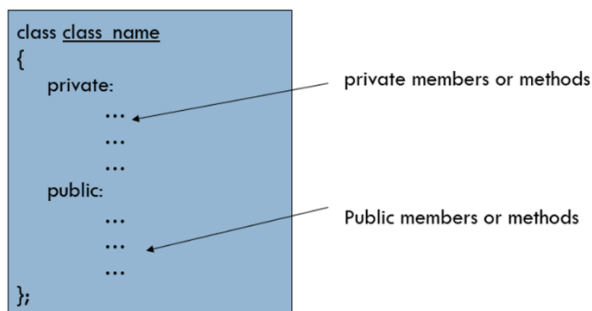
C++ supports three access specifiers:

- public
- private
- protected

The public access specifier allows a class to subject its member variables and member functions to other functions and objects.

The private access specifier allows a class to hide its member variables and member functions from other class objects and functions. (Default Access Specifier)

The protected access specifier allows a class to hide its member variables and member functions from other class objects and functions



Methods definition

The member function of the class can be defined in two different ways:

- **Inside the class definition:** - The member functions are simply defined inside the class only i.e.; the body of the function resides inside the range of class only.
- **Outside the class definition:** by using scope resolution operator, which specifies that the scope of the function is restricted to the class class_name.
Syntax: - class_name:: function_name

Inheritance

This mechanism of deriving a new class from existing/old class is called “inheritance”.

- The old class is known as “base” class, “super” class or “parent” class”.
- The new class is known as “sub” class “derived” class, or “child” class.

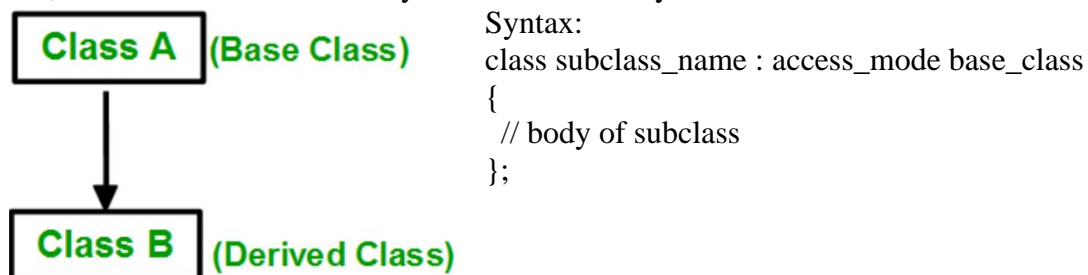
Syntax:

```
class <derived_class_name> : <access-specifier> <base_class_name>
{
    //body
}
```

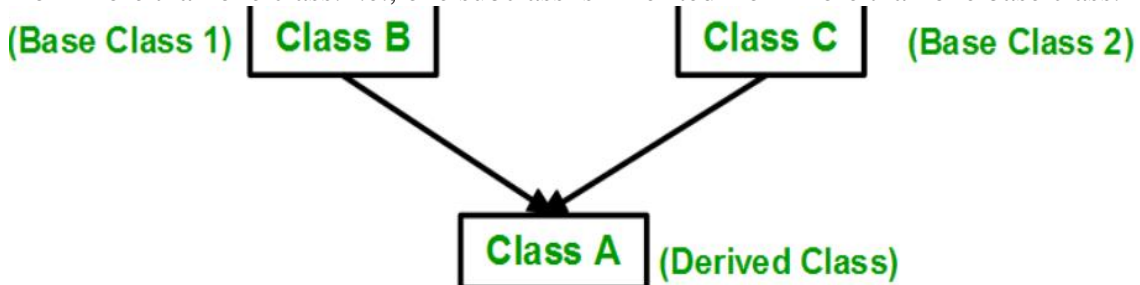
Types Of Inheritance: -

- Single inheritance
- Multilevel inheritance
- Multiple inheritance
- Hierarchical inheritance
- Hybrid inheritance

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.



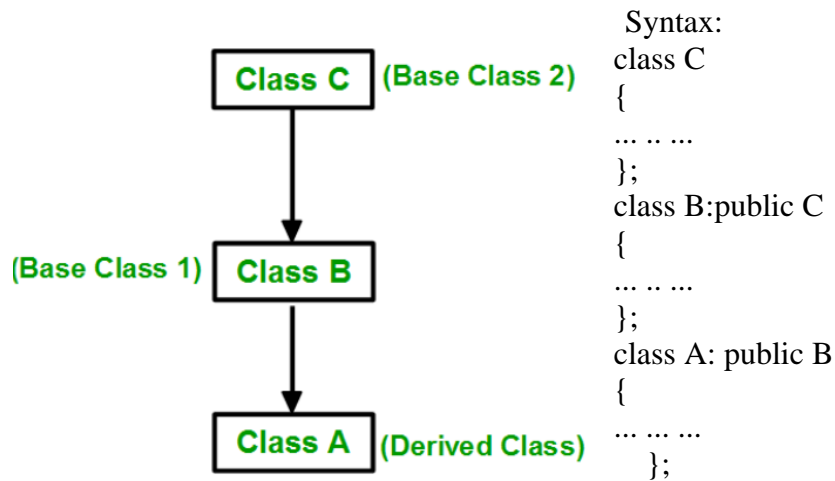
Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e., one subclass is inherited from more than one base class.



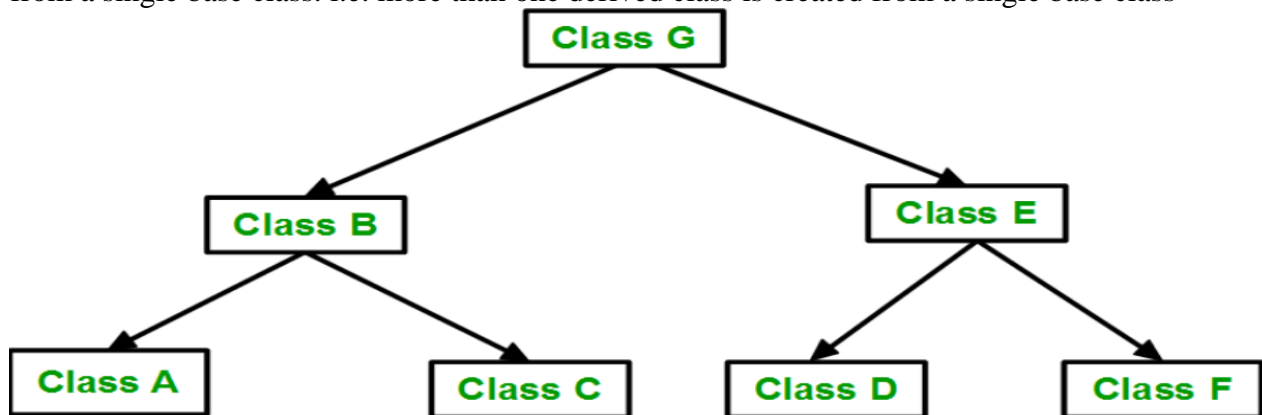
Syntax:

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
    // body of subclass
};
```

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



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

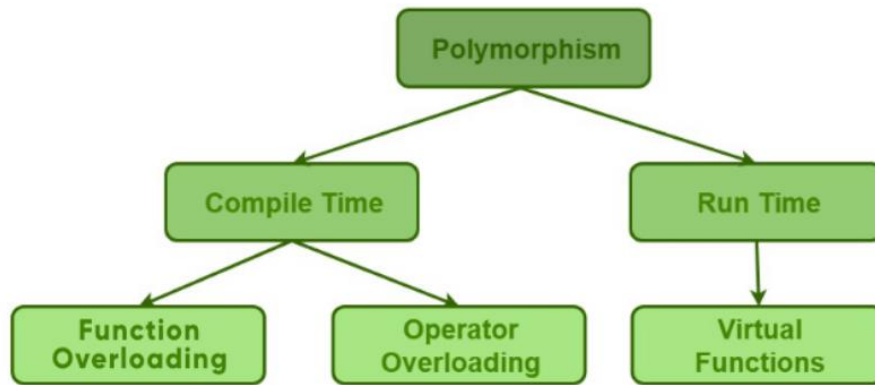


Syntax:

```
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

Polymorphism in C++

The word “polymorphism” means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. A real-life example of polymorphism is a person who at the same time can have different characteristics. Like a man at the same time is a father, a husband and an employee. So, the same person exhibits different behaviour in different situations. This is called polymorphism



Function Overloading: When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded. Functions can be overloaded by changing the number of arguments or/and changing the type of arguments.

Operator Overloading: We can make operators work for user-defined classes. This means the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

Virtual Function: A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

Encapsulation in C++:

In normal terms 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 manipulates them. Encapsulation also led to data abstraction or hiding. As using encapsulation also hides the data.

Encapsulation in C++



Abstraction in C++:

Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details.

Exception Handling

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords:

- throw – A program throws an exception when a problem shows up. This is done using a throw keyword.
- catch – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- try – A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Syntax:

```
try {  
    // protected code  
} catch (ExceptionName e1) {  
    // catch block  
} catch (Exception Name e2) {  
    // catch block  
} catch (ExceptionName eN) {  
    // catch block  
}
```

File Handling

File handling is used for store a data permanently in computer. Using file handling we can store our data in secondary memory.

For achieving file handling, we need to follow the following steps: -

STEP 1-Naming a file

STEP 2-Opening a file

STEP 3-Writing data into the file

STEP 4-Reading data from the file

STEP 5-Closing a file.

Classes for File stream operations: -

- ios
- istream
- ostream
- streambuf
- fstreambase
- ifstream
- ofstream
- fstream
- filebuf

DATA STRUCTURE:

A data structure is not only used for organizing the data. It is also used for processing, retrieving, and storing data. There are different basic and advanced types of data structures that are used in almost every program or software system that has been developed. So we must have good knowledge about data structures.

Classification of Data Structure:

➤ **Linear data structure:**

Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.

➤ **Static Data Structure:**

Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.

Ex- **Array**

➤ **Dynamic Data Structure:**

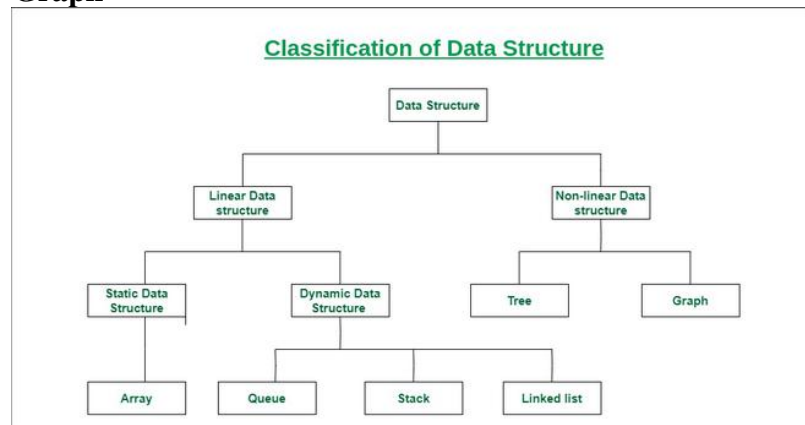
In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.

- **Queue**
- **Stack**
- **Linked list**

➤ **Non-linear data structure:**

Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.

- **Tree**
- **Graph**



Asymptotic Analysis

The efficiency of an algorithm depends on the amount of time, storage and other resources required to execute the algorithm. The efficiency is measured with the help of asymptotic notations. An algorithm may not have the same performance for different types of inputs. With the increase in the input size, the performance will change. The study of change in performance of the algorithm with the change in the order of the input size is defined as asymptotic analysis.

Asymptotic Notations:

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

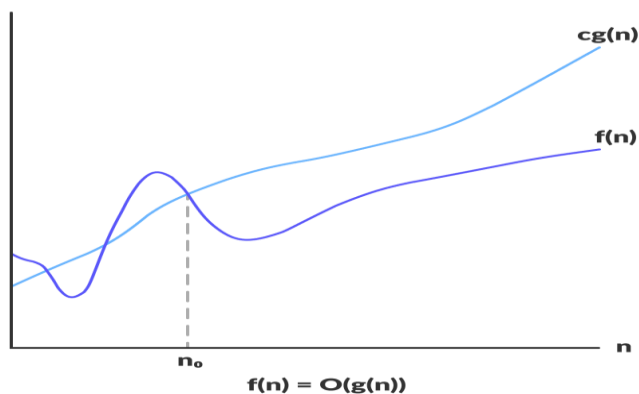
For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case. But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the worst case. When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

There are mainly three asymptotic notations:

- Big-O notation
- Omega notation
- Theta notation

Big-O Notation (O-notation):

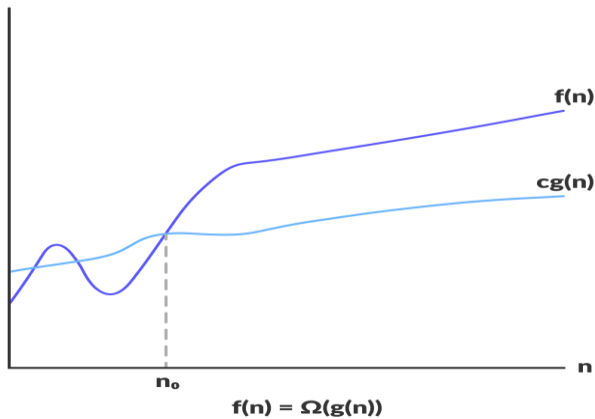
Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.



$$O(g(n)) = \{ f(n): \text{there exist positive constants } c \text{ and } n_0 \\ \text{such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$$

Omega Notation (Ω -notation):

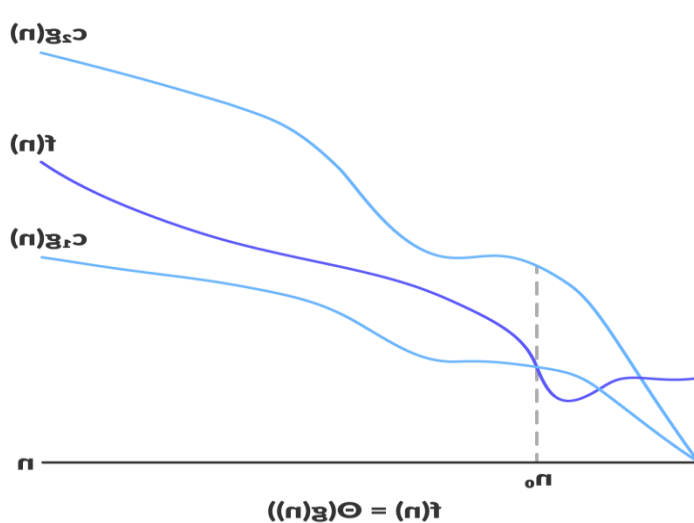
Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm.



$$\Omega(g(n)) = \{ f(n): \text{there exist positive constants } c \text{ and } n_0 \\ \text{such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$$

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 analyzing the average-case complexity of an algorithm.



$$\Theta(g(n)) = \{ f(n): \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \\ \text{such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0 \}$$

Divide and Conquer Algorithm:

A divide and conquer algorithm is a strategy of solving a large problem by

- breaking the problem into smaller sub-problems
- solving the sub-problems, and
- combining them to get the desired output.

To use the divide and conquer algorithm, recursion is used.

How Divide and Conquer Algorithms Work:

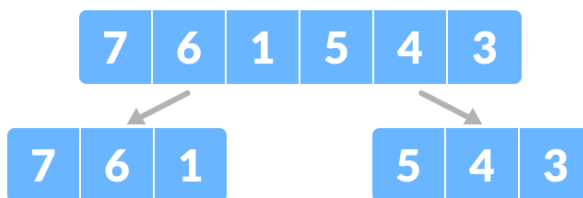
- Divide: Divide the given problem into sub-problems using recursion.
- Conquer: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly.
- Combine: Combine the solutions of the sub-problems that are part of the recursive process to solve the actual problem.

Let us understand this concept with the help of an example:

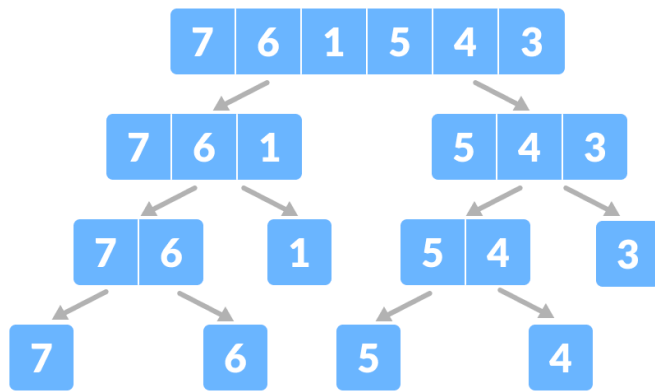
Let the given array be:



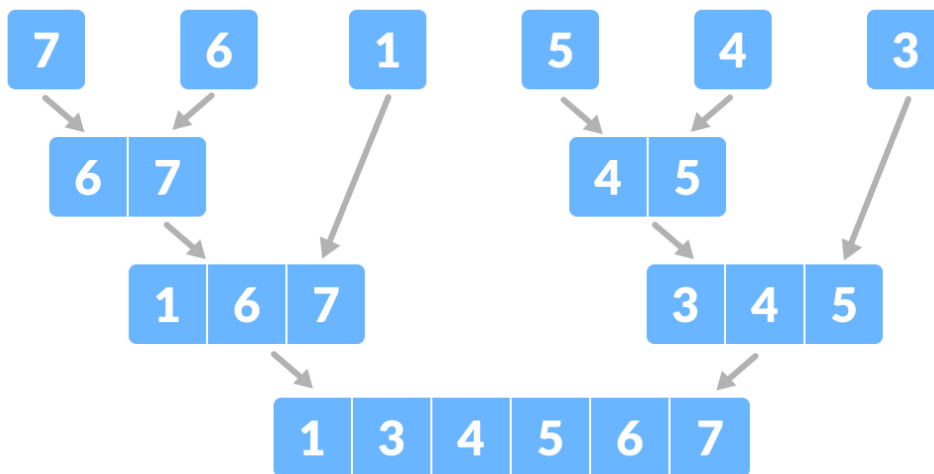
Divide the array into two halves



Again, divide each subpart recursively into two halves until you get individual elements.



Now, combine the individual elements in a sorted manner. Here, **conquer** and **combine** steps go side by side.



The complexity of the divide and conquer algorithm is calculated using the master theorem.

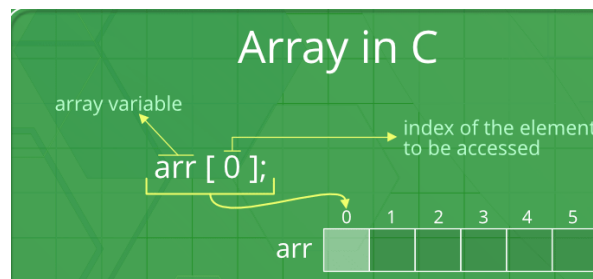
Array

An array is a collection of items of same data type stored at contiguous memory locations. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). The base value is index 0 and the difference between the two indexes is the offset.

Memory Location									
200	201	202	203	204	205	206	▪	▪	▪
U	B	F	D	A	E	C	▪	▪	▪
0	1	2	3	4	5	6	▪	▪	▪
Index									

Types of indexing in an array:

- 0 (zero-based indexing): The first element of the array is indexed by a subscript of 0.
- 1 (one-based indexing): The first element of the array is indexed by the subscript of 1.
- n (N-based indexing): The base index of an array can be freely chosen. Usually, programming languages allowing n-based indexing also allow negative index values, and other scalar data types like enumerations, or characters may be used as an array index.



How an Array is initialized?

Passing no value within the initializer: One can initialize the array by defining the size of the array and passing no values within the initializer.

Syntax: `int arr [5] = { };`

By passing specific values within the initializer: One can initialize the array by defining the size of the array and passing specific values within the initializer.

Syntax: `int arr [5] = { 1 , 2 , 3 , 4 , 5 };`

By passing specific values within the initializer but not declaring the size: One can initialize the array by passing specific values within the initializer and not particularly mentioning the size, the size is interpreted by the compiler.

Syntax: `int arr[] = { 1 , 2 , 3 , 4 , 5 };`

Universal Initialization: After the adoption of universal initialization in C++, one can avoid using the equals sign between the declaration and the initializer.

Syntax: `int arr[] { 1 , 2 , 3 , 4 , 5 };`

Operations on Array:

1. Insertion in Array:

We try to insert a value to a particular array index position, as the array provides random access it can be done easily using the assignment operator.

Pseudo Code: // to insert a value= 10 at index position 2;

`arr[2] = 10;`

Time Complexity:

- $O(1)$ to insert a single element
- $O(N)$ to insert all the array elements [where N is the size of the array]

2. Access elements in Array:

Accessing array elements become extremely important, in order to perform operations on arrays.

Pseudo Code: // to access array element at index position 2, we simply can write

`return arr[2] ;`

Time Complexity:

- $O(1)$

3. Searching in Array:

We try to find a particular value in the array, in order to do that we need to access all the array elements and look for the particular value.

Pseudo Code:// searching for value 2 in the array;

Loop from $i = 0$ to 5:

 check if `arr[i] = 2:`

 return true;

Time Complexity:

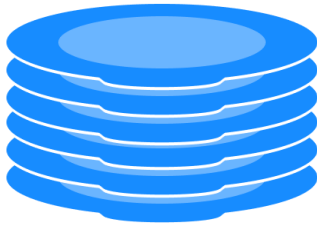
- $O(N)$, where N is the size of the array.

Types of arrays :

- One dimensional array (1-D arrays)
- Multidimensional array

Stack Data Structure:

A stack is a linear data structure that follows the principle of Last In First Out (LIFO). This means the last element inserted inside the stack is removed first.



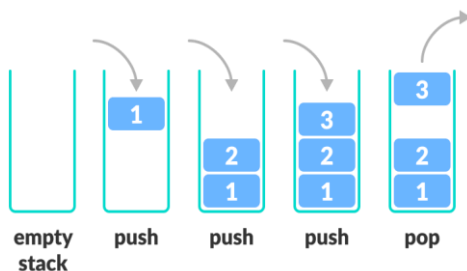
Stack representation similar to a pile of plate

Here, you can:

- Put a new plate on top
- Remove the top plate
- And, if you want the plate at the bottom, you must first remove all the plates on top.
This is exactly how the stack data structure works.

LIFO Principle of Stack:

In programming terms, putting an item on top of the stack is called push and removing an item is called pop.



In the above image, although item 3 was kept last, it was removed first. This is exactly how the LIFO (Last In First Out) Principle works.

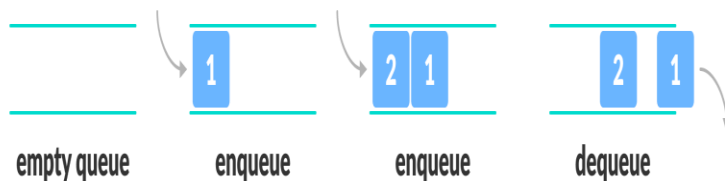
Basic Operations of Stack:.

- Push: Add an element to the top of a stack
- Pop: Remove an element from the top of a stack
- IsEmpty: Check if the stack is empty
- IsFull: Check if the stack is full
- Peek: Get the value of the top element without removing it

Queue Data Structure:

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue follows the First In First Out (FIFO) rule - the item that goes in first is the item that comes out first.



In the above image, since 1 was kept in the queue before 2, it is the first to be removed from the queue as well. It follows the FIFO rule.

In programming terms, putting items in the queue is called enqueue, and removing items from the queue is called dequeue.

Basic Operations of Queue:

A queue is an object (an abstract data structure - ADT) that allows the following operations:

- Enqueue: Add an element to the end of the queue
- Dequeue: Remove an element from the front of the queue
- IsEmpty: Check if the queue is empty
- IsFull: Check if the queue is full
- Peek: Get the value of the front of the queue without removing it.

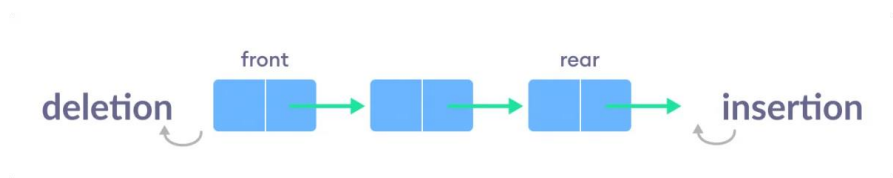
Types of Queues

There are four different types of queues:

- Simple Queue
- Circular Queue
- Priority Queue
- Double Ended Queue

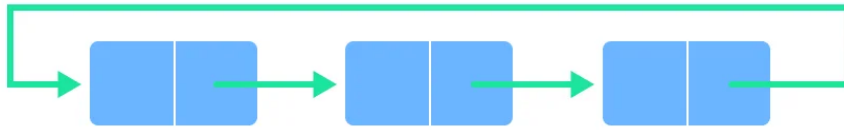
Simple Queue

In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule



Circular Queue

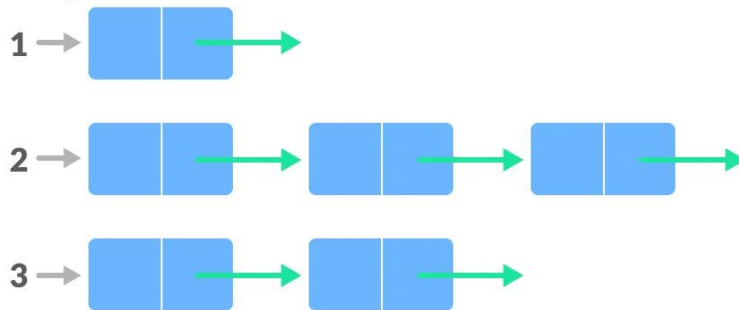
In a circular queue, the last element points to the first element making a circular link.



Priority Queue

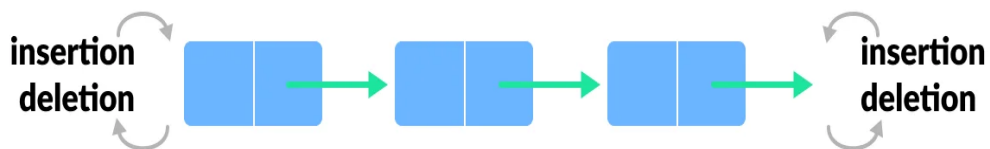
A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.

Priority



Deque (Double Ended Queue)

In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.



Linked list Data Structure:

A linked list is a linear data structure that includes a series of connected nodes. Here, each node stores the data and the address of the next node. For example,



You have to start somewhere, so we give the address of the first node a special name called Head. Also, the last node in the linked list can be identified because its next portion points to Null. Linked lists can be of multiple types: **singly, doubly, and circular linked list.**

Representation of Linked List:

Each node consists:

- A data item
- An address of another node

struct node

```
{  
    int data;  
    struct node *next;  
};
```

Each struct node has a data item and a pointer to another struct node.



Linked List Complexity:

Time Complexity:

Worst case	Average Case	
Search	O(n)	O(n)
Insert	O(1)	O(1)

Deletion	O(1)	O(1)
----------	------	------

Space Complexity: $O(n)$

Linked List Applications:

- Dynamic memory allocation
- Implemented in stack and queue
- In **undo** functionality of softwares
- Hash tables, Graphs

Linked List Operations:

There are various linked list operations that allow us to perform different actions on linked lists. For example, the insertion operation adds a new element to the linked list.

- Basic linked list operations:
- TRAVERSAL- access each element of the linked list
- INSERTION- adds a new element to the linked list
- DELETION - removes the existing elements
- SEARCH - find a node in the linked list
- SORT- sort the nodes of the linked list

Types of Linked List –

- Singly linked
- doubly linked
- circular

Singly Linked List



Doubly Linked List



Circular Linked List



Hash Table:

The Hash table data structure stores elements in key-value pairs where

- Key- unique integer that is used for indexing the values
- Value - data that are associated with keys.

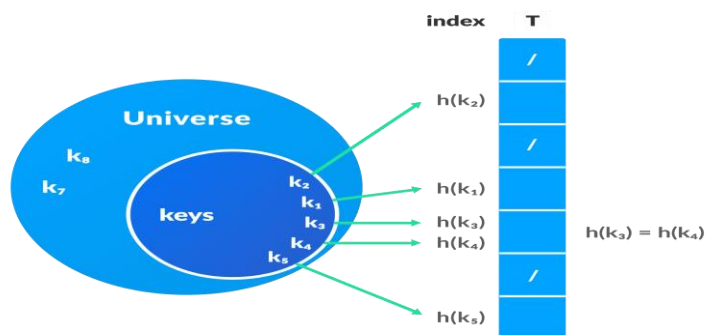


Hashing (Hash Function):

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called hashing.

Let k be a key and $h(x)$ be a hash function.

Here, $h(k)$ will give us a new index to store the element linked with k .



Hash Collision

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a **hash collision**.

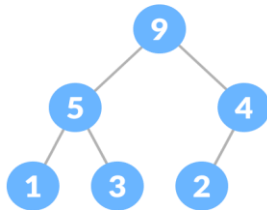
We can resolve the hash collision using one of the following techniques.

- Collision resolution by chaining
- Open Addressing: Linear/Quadratic Probing and Double Hashing

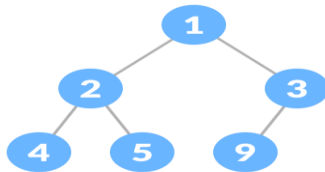
Heap Data Structure:

Heap data structure is a complete binary tree that satisfies the heap property, where any given node is

- always greater than its child node/s and the key of the root node is the largest among all other nodes. This property is also called max heap property.
- always smaller than the child node/s and the key of the root node is the smallest among all other nodes. This property is also called min heap property.



Max-heap



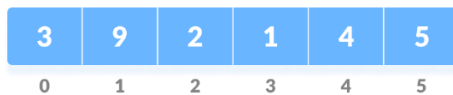
Min-heap

Heap Operations:

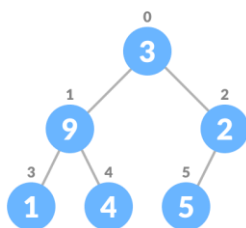
Heapify:

Heapify is the process of creating a heap data structure from a binary tree. It is used to create a Min-Heap or a Max-Heap.

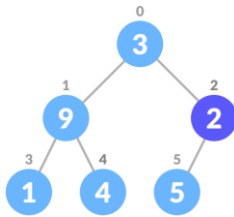
Let the input array be;



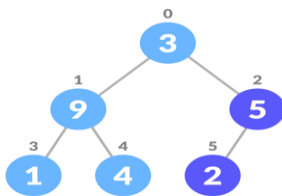
Create a complete binary tree from the array:



Start from the first index of non-leaf node whose index is given by $n/2 - 1$.



- Set current element i as **largest**:
- The index of left child is given by $2i + 1$ and the right child is given by $2i + 2$. If **leftChild** is greater than **currentElement** (i.e. element at i th index), set **leftChildIndex** as **largest**. If **rightChild** is greater than element in **largest**, set **rightChildIndex** as **largest**.
- Swap **largest** with **currentElement**.



- Repeat steps 3-7 until the subtrees are also heapified.

Fibonacci Heap:

A fibonacci heap is a data structure that consists of a collection of trees which follow min heap or max heap property. We have already discussed min heap and max heap property in the Heap Data Structure article. These two properties are the characteristics of the trees present on a fibonacci heap.

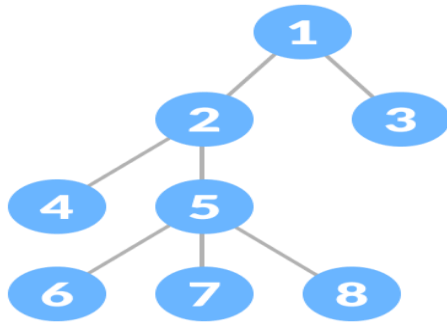
In a fibonacci heap, a node can have more than two children or no children at all. Also, it has more efficient heap operations than that supported by the binomial and binary heaps. The fibonacci heap is called a fibonacci heap because the trees are constructed in a way such that a tree of order n has at least F_{n+2} nodes in it, where F_{n+2} is the $(n + 2)$ th Fibonacci number.

Properties of a Fibonacci Heap:

- It is a set of min heap-ordered trees. (i.e. The parent is always smaller than the children.)
- A pointer is maintained at the minimum element node.
- It consists of a set of marked nodes. (Decrease key operation)

- **Tree Data Structure:**

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



Why Tree Data Structure:

Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But, it is not acceptable in today's computational world. Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.

Types of Tree:

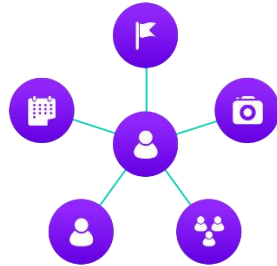
- Binary Tree
- Binary Search Tree
- AVL Tree
- B-Tree

Tree Applications:

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Graph Data Structure:

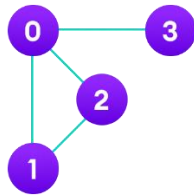
A graph data structure is a collection of nodes that have data and are connected to other nodes. Every relationship is an edge from one node to another.



More precisely, a graph is a data structure (V, E) that consists of

A collection of vertices V .

A collection of edges E , represented as ordered pairs of vertices (u,v) .



In the graph,

$V = \{0, 1, 2, 3\}$, $E = \{(0,1), (0,2), (0,3), (1,2)\}$, $G = \{V, E\}$

Graph Terminology

- Adjacency: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
- Path: A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
- Directed Graph: A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

PROJECT UNDERTAKEN

Introduction Of Payroll Management System Project

A payroll management system is a software that is used to manage all your employee's financial records in a simple and automated fashion. This payroll management system manages employee's salaries, deductions, other conveyance, net pay, etc.

Payroll Management Software eases the hassles of managing vast amounts of data by making the employee or faculty management & payroll processing easier. It allows the administrator to pay on-time & accurate amounts of salary to all the employees working in the educational institution.

Project Work

The payroll management system starts by creating a new id with name and password given by the user. After creating a new id it will lead user to a login page.

After successful login the user will have set of operations like, insert, delete, search, update, etc. The user has the free will to choose any operation provided by the payroll management system.

If the user wishes to insert a new data they need to provide the employee's name and id. If the user wants to delete, they can do so by providing the employee's id.

After manipulating and making sure all data provided is accurate the system will show the accurate salary of the employee whose id was provided by the user. At the end the user can exit the payroll management system.

OBJECTIVE

The main objective of payroll management system is to ease the efforts of the institution in calculating the accurate salaries and give it to the employees on time.

The payroll management system helps administrator to easily insert, delete, search or/and calculating salaries of large number of employees working in the institution without spending much time as compared to doing it manually

Conclusion

Data structure are the great tool to computer science and professionals who utilize them. This course covered the basics of data structures. With this I have only scratched the surface. Although I have built a good foundation to move ahead. Data structure is not just limited to Stack, Queues and linked lists but is quit a vast area. There are many more data structure which include Maps, Hash table, Graphs, Trees etc. Each data structure has its own advantage and disadvantage and must be used according to the need of application. Many high levels and object-oriented programming language like c#, java, python, come built in with many of these data structures, therefore it is important. While doing this summer internship I have learned many new things and it was a wonderful experience for me. This course has developed my thinking skills, problem solving skills and many more. Overall, it was beneficial for me and hope that my project would be interesting and ever knowledgeable.

BIBLIOGRAPHY

- **Content shared by Fifth Force**
- **Tutorialpoint website**
- **Geeks for Geeks website**