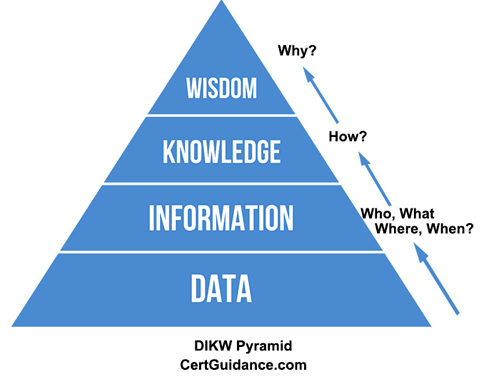
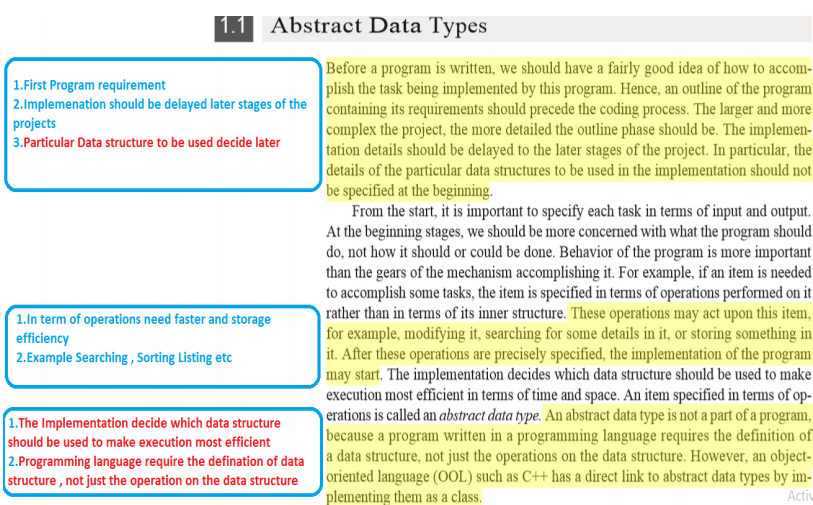
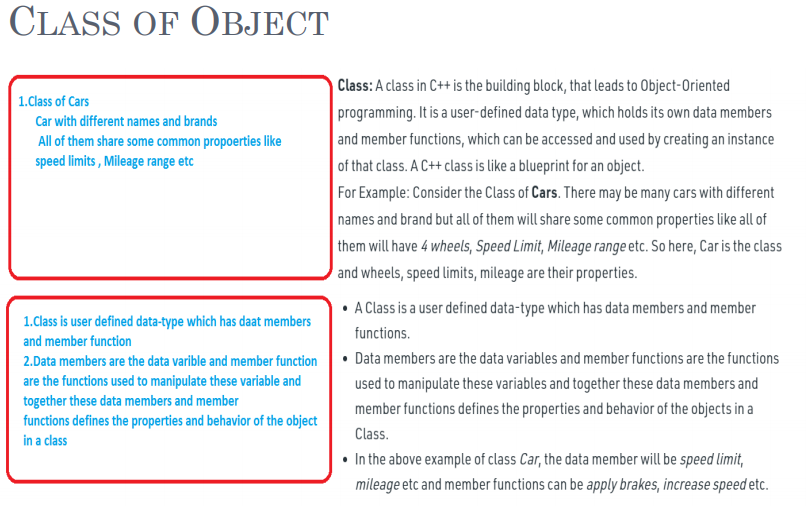
**Lecture -2**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++





**Four Pillars of OOPS**

[Encapsulation](https://www.geeksforgeeks.org/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 manipulate them.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data.

Sample Code-1

#include<iostream>

using namespace std;

class Encapsulation

{

    private:

        // data hidden from outside world

        int x;

    public:

        // function to set value of

        // variable x

        void set(int a)

        {

            x =a;

        }

        // function to return value of

        // variable x

        int get()

        {

            return x;

        }

};

// main function

int main()

{

    Encapsulation obj;

    obj.set(5);

    cout<<obj.get();

    return 0;

}

Inportant Point Discussion “cout <<"Size of Empty Class is = "<< sizeof(Empty\_class);”

[Abstraction](https://www.geeksforgeeks.org/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. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

Sample code-2

#include <iostream>

using namespace std;

class implementAbstraction

{

    private:

        int a, b;

    public:

        // method to set values of

        // private members

        void set(int x, int y)

        {

            a = x;

            b = y;

        }

        void display()

        {

            cout<<"a = " <<a << endl;

            cout<<"b = " << b << endl;

        }

};

int main()

{

    implementAbstraction obj;

    obj.set(10, 20);

    obj.display();

    return 0;

}

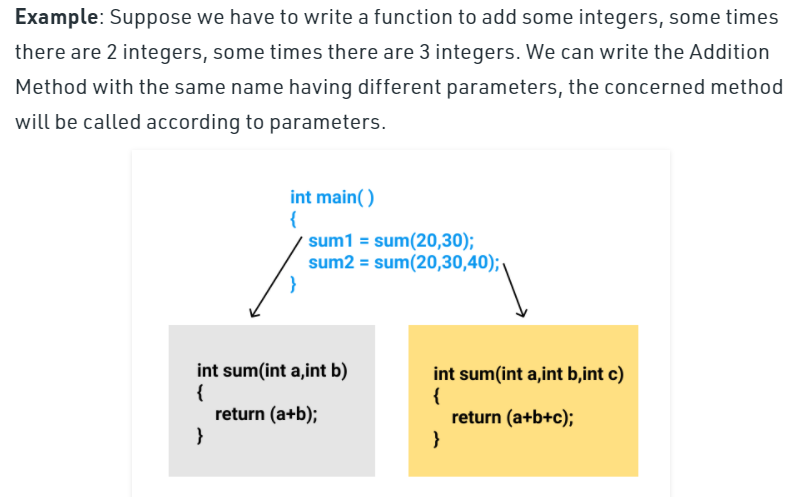
[Polymorphism:](https://www.geeksforgeeks.org/polymorphism-in-c/) The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

C++ supports operator overloading and function overloading.

* *Operator Overloading*: The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
* *Function Overloading*: Function overloading is using a single function name to perform different types of tasks.  
  Polymorphism is extensively used in implementing inheritance.



// Sample code-3

#include <bits/stdc++.h>

using namespace std;

class Geeks

{

    public:

    // function with 1 int parameter

    void func(int x)

    {

        cout << "value of x is " << x << endl;

    }

    // function with same name but 1 double parameter

    void func(double x)

    {

        cout << "value of x is " << x << endl;

    }

    // function with same name and 2 int parameters

    void func(int x, int y)

    {

        cout << "value of x and y is " << x << ", " << y << endl;

    }

};

int main() {

    Geeks obj1;

    // Which function is called will depend on the parameters passed

    // The first 'func' is called

    obj1.func(7);

    // The second 'func' is called

    obj1.func(9.132);

    // The third 'func' is called

    obj1.func(85,64);

    return 0;

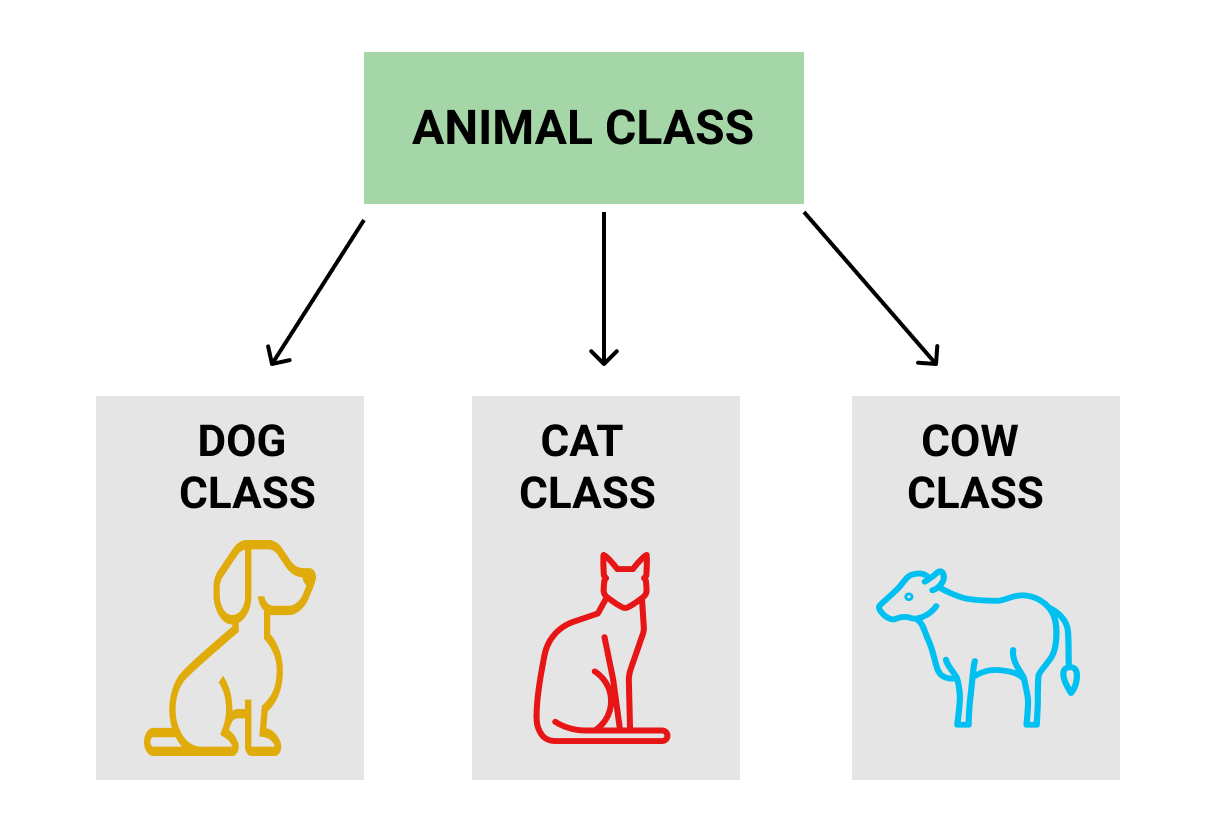
}

**Note :Disscuss More in Next Class**

[Inheritance](https://www.geeksforgeeks.org/inheritance-in-c/): The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

* **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class.
* **Super Class**:The class whose properties are inherited by sub class is called Base Class or Super class.
* **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**Example**: Dog, Cat, Cow can be Derived Class of Animal Base Class.



// Sample code-4

// of Inheritance

#include <bits/stdc++.h>

using namespace std;

//Base class

class Parent

{

public:

int id\_p;

};

// Sub class inheriting from Base Class(Parent)

class Child : public Parent

{

public:

int id\_c;

};

//main function

int main()

{

Child obj1;

// An object of class child has all data members

// and member functions of class parent

obj1.id\_c = 7;

obj1.id\_p = 91;

cout << "Child id is " << obj1.id\_c << endl;

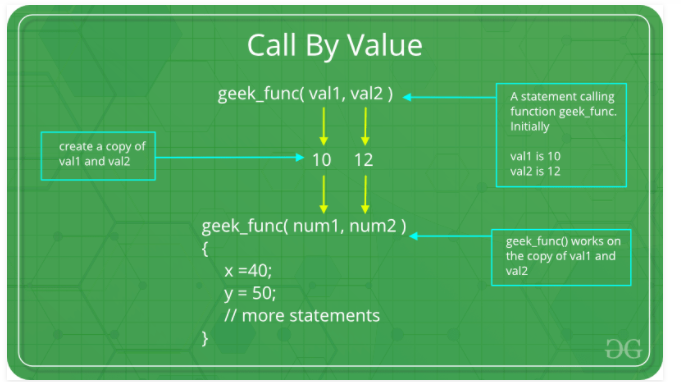
cout << "Parent id is " << obj1.id\_p << endl;

return 0;

}

Again Discuss More in Next Class

**Functions and parameter passing**



// C program to illustrate

// call by value

#include <stdio.h>

void func(int a, int b)

{

    a += b;

    printf("In func, a = %d b = %d\n", a, b);

}

int main(void)

{

    int x = 5, y = 7;

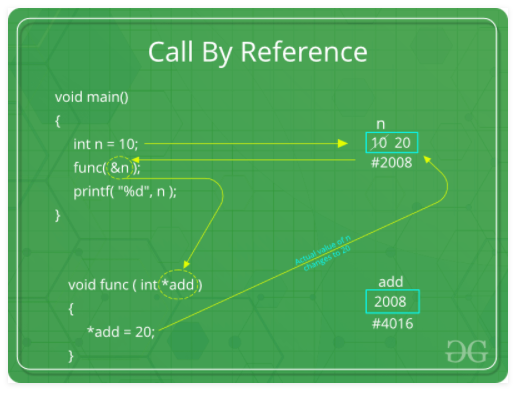
    // Passing parameters

    func(x, y);

    printf("In main, x = %d y = %d\n", x, y);

    return 0;

}



// C program to illustrate

// call by reference

#include <stdio.h>

void swapnum(int\* i, int\* j)

{

int temp = \*i;

\*i = \*j;

\*j = temp;

}

int main(void)

{

int a = 10, b = 20;

// passing parameters

swapnum(&a, &b);

printf("a is %d and b is %d\n", a, b);

return 0;

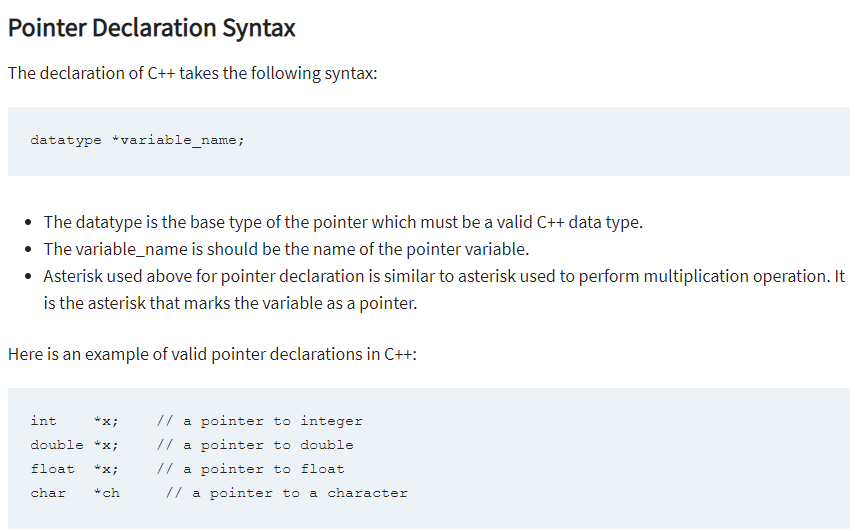
}

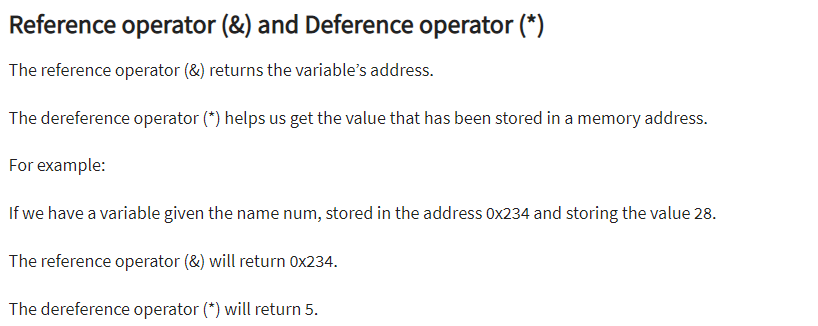
**Other methods of Parameter Passing**

These techniques are older and were used in earlier programming languages like Pascal, Algol and Fortran. These techniques are not applicable in high level languages.

1. **Pass by Result :**This method uses *out-mode* semantics. Just before control is transfered back to the caller, the value of the formal parameter is transmitted back to the actual parameter.T his method is sometimes called *call by result*. In general, pass by result technique is implemented by copy.
2. **Pass by Value-Result :** This method uses *in/out-mode* semantics. It is a combination of Pass-by-Value and Pass-by-result. Just before the control is transferred back to the caller, the value of the formal parameter is transmitted back to the actual parameter. This method is sometimes called as *call by value-result*
3. **Pass by name :**This technique is used in programming language such as ***Algol***. In this technique, symbolic “*name*” of a variable is passed, which allows it both to be accessed and update.

**Pointers and their uses**





include <iostream>

using namespace std;

int main() {

int x = 27;

int \*ip;

ip = &x;

cout << "Value of x is : ";

cout << x << endl;

cout << "Value of ip is : ";

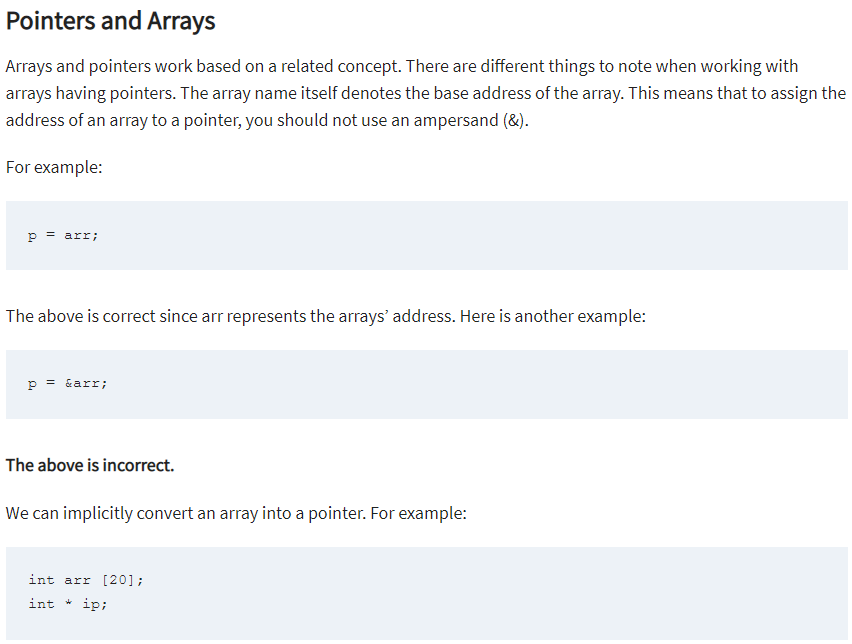
cout << ip<< endl;

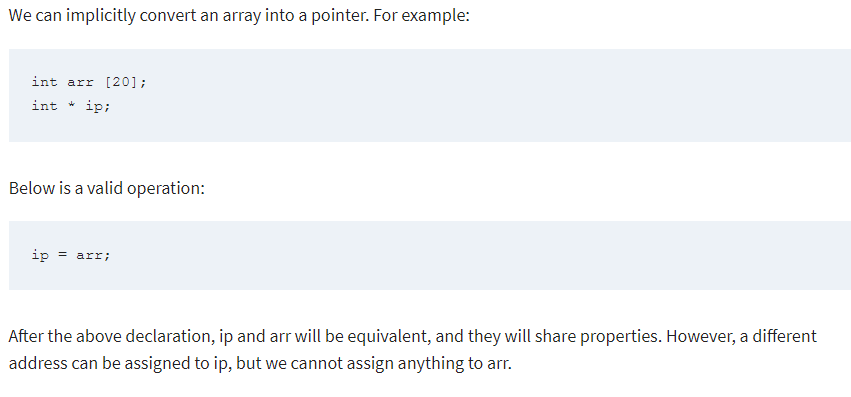
cout << "Value of \*ip is : ";

cout << \*ip << endl;

return 0;

}





#include <iostream>

using namespace std;

int main() {

int \*ip;

int arr[] = { 10, 34, 13, 76, 5, 46 };

ip = arr;

for (int x = 0; x < 6; x++) {

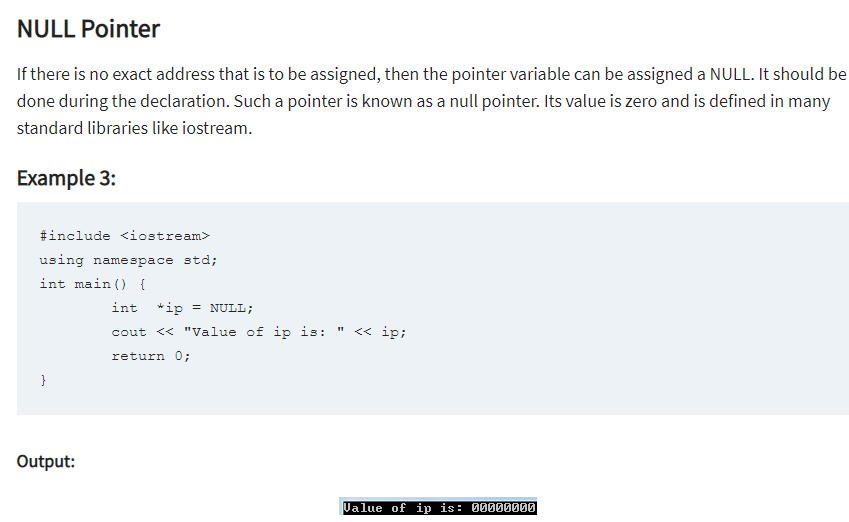
cout << \*ip << endl;

ip++;

}

return 0;

}



**Pointers of Variables**

With C++, you can manipulate data directly from the computer’s memory.

The memory space can be assigned or re-assigned as one wishes. This is made possible by Pointer variables.

Pointer variables point to a specific address in the computer’s memory pointed to by another variable.

It can be declared as follows:

int \*p;

Or,

int\* p;

In the you example, we have declared the pointer variable p.

It will hold a memory address.

The asterisk is the dereference operator that means a pointer to.

The pointer p is pointing to an integer value in the memory address.

#include <iostream>

using namespace std;

int main() {

int \*p, x = 30;

p = &x;

cout << "Value of x is: " << \*p;

return 0;

}

## Application of Pointers

Functions in C++ can return only one value. Further, all the variables declared in a function are allocated on the function call stack. As soon as the function returns, all the stack variables are destroyed.

Arguments to function are passed by value, and any modification done on the variables doesn’t change the value of the actual variables that are passed. Following example helps illustrate this concept:-

#include <iostream>

using namespace std;

void test(int\*, int\*);

int main() {

int a = 5, b = 5;

cout << "Before changing:" << endl;

cout << "a = " << a << endl;

cout << "b = " << b << endl;

test(&a, &b);

cout << "\nAfter changing" << endl;

cout << "a = " << a << endl;

cout << "b = " << b << endl;

return 0;

}

void test(int\* n1, int\* n2) {

\*n1 = 10;

\*n2 = 11;

}