C++ Part 1

#include<iostream>

using namespace std;

main()

{

cout<<"Hello world";

}

* cout -> object
* << ->insertion operator

**namespace**

if we write a ,multiple line of codes the part of of code is slitted and named it, that's called namespace

same namespace group content we can access without providing the name , bit if we want to access from outside group we need to

mention the name

**namespace std ;** has a definition about **"cout**" that why we writing namespace here , and **cout is** called as **object** and it's from

**namespace std class**

**scope resolution operator**

using namespace std;

we can include namespace std and we can use cout another way is we can use :: operator

eg:

std::cout -->we can use this to avoid including std

Helps when you are created same name object from diffrent class , let say i created cout in abc class and by default std has cout , if i use cout by including namespace std and namespace abc , it will be a problem. In this case we can use :: to explain it’s belongs to some specific class.

**New line**

endl

cout << “Hello world”<<endl<<”Welcome”;

endl also came from namespace std , so we have to include.

**How to print value of variable**

int value=20;

cout<<”Value is ”<<value;

How to read input from user

int data1, data2;

cout<< ”Enter the data1 and data2 ”<<endl;

cin>>data1>>data2;

cout<<”The data1 is ”<<endl<<data1<<”The data2 is ”<<endl<<data2;

OOPs concept

### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

### Class

Collection of objects is called class. It is a logical entity.

### Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

### Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

* Class and object is similar to structure

#include <iostream>  
using namespace std;   
class student{  
 public:  
 int id;  
 string name;  
};

int main()  
{  
 student s1; // creating object for class, when ever we create an object the memory will be allocated for entier objects  
 s1.id=200;// accessing the members of class using object and assigning value  
 s1.name="Aravinth";  
 cout<<s1.id<<endl<<s1.name<<endl;  
 return 0;  
}

**Methods**

#include <iostream>   
 using namespace std;   
 class Student {   
 public:   
 int id;//data member (also instance variable)   
 string name;//data member(also instance variable)   
 void insert(int i, string n) // class with "method type object"  
 {   
 id = i;   
 name = n;   
 }   
 void display()   
 {   
 cout<<id<<" "<<name<<endl;   
 }   
 };   
 int main(void) {   
 Student s1; //creating an object of Student when ever we create a object the memory will be allocated for entier objects  
 Student s2; //creating an object of Student   
 s1.insert(201, "Sonoo"); //we can call the method of class similar to variable member  
 s2.insert(202, "Nakul");   
 s1.display();   
 s2.display();   
 return 0;   
 }

Method with parameter with return

#include <iostream>   
 using namespace std;   
 class Employee {   
 public:   
 int id;//data member (also instance variable)   
 string name;//data member(also instance variable)   
 float salary;   
 int insert(int i, string n, float s)   
 {   
 id = i;   
 name = n;   
 salary = s;  
 i=i+s;  
 return i;  
 }   
 void display()   
 {   
 cout<<id<<" "<<name<<" "<<salary<<endl;   
 }   
 };   
 int main(void) {   
 Employee e1; //creating an object of Employee   
 Employee e2; //creating an object of Employee   
 int result1=e1.insert(201, "Sonoo",9);   
 int result2=e2.insert(202, "Nakul", 2);   
 cout<<result1<<endl<<result2<<endl;  
 e1.display();   
 e2.display();   
 return 0;   
 }

**Constructor**

It’s a special method which is invoked automatically at the time of object creation.

Two types of constructors in C++.

* Default constructor
* Parametrized constructor

## **C++ Default Constructor**

A constructor which has no argument is known as default constructor.

#include <iostream>   
 using namespace std;   
 class Employee   
 {   
 public:   
 Employee() // Constructor must be same name of the class   
 {   
 cout<<"Default Constructor Invoked"<<endl;   
 }   
 };   
 int main(void)   
 {   
 Employee e1; //creating an object of Employee during this line the Constructor automatically called   
 Employee e2;   
 return 0;   
 }

## **C++Parametrized Constructor**

#include <iostream>

using namespace std;

class Employee {

public:

int id;//data member (also instance variable)

string name;//data member(also instance variable)

float salary;

Employee(int i, string n, float s)

{

id = i;

name = n;

salary = s;

}

void display()

{

cout<<id<<" "<<name<<" "<<salary<<endl;

}

};

int main(void) {

Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee and we passing the parameter by mentioning the constructor name

Employee e2=Employee(102, "Nakul", 59000);

e1.display();

e2.display();

return 0;

}

## **C++ Constructor and Destructor Example**

#include <iostream>   
 using namespace std;   
 class Employee   
 {   
 public:   
 Employee()   
 {   
 cout<<"Constructor Invoked"<<endl;   
 }   
 ~Employee() // destructor must be same name as class but ~ symbol will be mentioned before, destructor will be called before return 0 that is end of program   
 {   
 cout<<"Destructor Invoked"<<endl;   
 }   
 };   
 int main(void)   
 {   
 Employee e1; //creating an object of Employee   
 Employee e2; //creating an object of Employee   
 int a=20;  
 cout<<"Test string"<<endl<<a<<endl;  
 return 0;   
 }

Constructor Invoked

Constructor Invoked

Test string

20

Destructor Invoked

Destructor Invoked

# **C++ this Pointer**

this is a pointer which points address

which address?

While creating a object (in this case Example e1;) the this pointer holds the address of object called e1

#include <iostream>   
 using namespace std;

class Example   
 {   
 int a;// private variable only accesed by within a class  
 public:   
 Example()   
 {   
 cout<<"Constructor Invoked"<<endl;   
 a=1;  
 }   
 void print()  
 {  
 cout<<"a="<<a<<endl; //directly we printing a here   
 cout<<"this->a="<<this->a<<endl; // we can also print a using this keyword by derefrencing the address  
 cout<<"(\*this).a="<<(\*this).a<<endl;  
 }  
 };   
 int main(void)   
 {   
 Example e1; //creating an object of Example, while creating the object the e1's address will be stored in this pointer  
 e1.print();  
 return 0;   
 }

**This pointer use cases**

#include <iostream>   
 using namespace std;   
 class Example   
 {   
 int a;// private variable only accesed by within a class  
 public:   
 Example()   
 {   
 cout<<"Constructor Invoked"<<endl;   
 a=1;  
 }   
 void print(int a)  
 {  
 cout<<"print functions a is "<<a<<endl; //directly we printing a here   
 cout<<"class a is "<<this->a<<endl;  
 a=a;  
 cout<<"a value is "<<a<<endl;  
 a=this->a;  
 cout<<"a value is "<<a<<endl;  
 cout<<"this->a="<<this->a<<endl; // we can also print a using this keyword by derefrencing the address  
 // cout<<"(\*this).a="<<(\*this).a<<endl;  
 }  
 };   
 int main(void)   
 {   
 Example e1; //creating an object of Example, while creating the object the e1's address will be stored in this pointer  
 e1.print(10);  
 return 0;   
 }

Constructor Invoked

print functions a is 10

class a is 1

a value is 10

a value is 1

**Static Field**

static is used to create a common member for all object.

If we create an object the memory is allocated

if we use static keyword for a member or variable, it can’t initialized again and again

whenever first object is created in parallel the static variable will be initialized

if again another object is created static variable will not initialized again but the variable without having a static keyword are initialized with respect to object

**Namespace**

* you might be writing some code that has a function called xyz() and there is another library available which is also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code.
* A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries.

**Syntax**

namespace namespace\_name {

// code declarations

}

/\*  
Note: In this program same name for two functions called func()  
but two function performs different operations  
we have to differentiate the function to compiler for that we are using namespace  
\*/  
#include <iostream>  
using namespace std;  
  
// first name space  
namespace first\_space {  
 void func() {   
 cout << "Inside first\_space" << endl;  
 }  
}  
  
// second name space  
namespace second\_space {  
 void func() {  
 cout << "Inside second\_space" << endl;  
 }  
}  
  
int main () {  
 // Calls function from first name space.  
 first\_space::func();  
   
 // Calls function from second name space.  
 second\_space::func();   
  
 return 0;  
}

we can also use using namespace to declare the namespace so that we can avoid the :: operator

#include <iostream>

using namespace std;

// first name space

namespace first\_space {

void func() {

cout << "Inside first\_space" << endl;

}

}

// second name space

namespace second\_space {

void func() {

cout << "Inside second\_space" << endl;

}

}

using namespace first\_space;

int main () {

// This calls function from first name space.

func();

return 0;

}

we can’t add two namespace together

eg:

using namespace first\_space;

using namespace second\_space;

if we try to call func() --> it will through error

# **C++ Access Specifiers**

Access specifiers define how the members (attributes and methods) of a class can be accessed.

In C++, there are three access specifiers:

* public - members are accessible from **outside the class**
* private - members **cannot be accessed** (or viewed) from **outside the class**
* protected - members **cannot be accessed** from **outside the class,** however, they can be accessed in inherited classes. You will learn more

Public – can be accessed from main function and everywhere

Private – we can access only inside a class using methods

Protected – members only accessable bu inherited classes

class MyClass {  
public: // Public access specifier  
 int x; // Public attribute  
private: // Private access specifier  
 int y; // Private attribute  
};  
  
int main() {  
 MyClass myObj;  
 myObj.x = 25;// Allowed (public)  
 myObj.y = 50; // Not allowed (private)  
 return 0;  
}

error: y is private

**Encapsulation**

we can access private members only inside a class, if we want to access it on main we have to implement methods inside a class which helps us to read and write a variable from main or outside of the class, its called encapsulation.

#include <iostream>  
using namespace std;  
  
class Adder {  
 public:  
 // constructor  
 Adder(int i = 0) {  
 total = i;  
 }  
   
 // interface to outside world  
 void addNum(int number) {  
 total += number;  
 }  
   
 // interface to outside world  
 int getTotal() {  
 return total;  
 };  
   
 private:  
 // hidden data from outside world  
 int total;  
};  
  
int main() {  
 Adder a;  
   
 a.addNum(10);// we can write a variable using the class method called addNum , so that we can assign value to total which is a private to class  
 a.addNum(20);  
 a.addNum(30);  
  
 cout << "Total " << a.getTotal() <<endl;// similarly we are reading the value here  
 return 0;  
}

Total 60

## **Inheritance**

When one object acquires all the properties and behaviours of parent object **i.e. known as inheritance.**

We can add a base class(first) to another base class(second) and we can directly access the first base class memeber by crating object of second base class, the second class also called as derived class or child class.

Class base

{

public:

int variable;

};

class child: public base

{

};

child obj;

obj.variable=20;// it’s possible

Eg:

/ Base class  
class Vehicle {  
 public:   
 string brand = "Ford";  
 void honk() {  
 cout << "Tuut, tuut! \n" ;  
 }  
};  
  
// Derived class  
class Car: public Vehicle {  
 public:   
 string model = "Mustang";  
};  
  
int main() {

Car myCar;// we crating derived class object

myCar.honk(); // we can access base class function using derived class object.  
 cout << myCar.brand + " " + myCar.model;  
 return 0;  
}

Types:

* Inheritance
* Multilevel Inheritance
* Multiple Inheritance

**Multilevel inheritance**

class A -> class A:B -> class C:B

class A --> only access members of A

class A:B--> can access class B and class A members bcz B having a reference of A

class C:B --> can access class C, B, A members

// Base class (parent)  
class MyClass {  
 public:   
 void myFunction() {  
 cout << "Some content in parent class." ;  
 }  
};  
  
// Derived class (child)  
class MyChild: public MyClass {  
};  
  
// Derived class (grandchild)   
class MyGrandChild: public MyChild {  
};  
  
int main() {  
 MyGrandChild myObj;

myObj.myFunction();  
 return 0;  
}

**Multiple inheritance**

A class can also be derived from more than one base class, using a comma-separated list:

Class A --> Base class only accessed by own members

Class B --> Base class only accessed by own members

Class C: public A, public B --> Class C can access both Class A and Class B members

// Base class  
class MyClass {

s public:   
 void myFunction() {  
 cout << "Some content in parent class." ;  
 }  
};  
  
// Another base class  
class MyOtherClass {  
 public:   
 void myOtherFunction() {  
 cout << "Some content in another class." ;  
 }  
};  
  
// Derived class   
class MyChildClass: public MyClass, public MyOtherClass {  
};  
  
int main() {  
 MyChildClass myObj;  
 myObj.myFunction();  
 myObj.myOtherFunction();

return 0;  
}

**References**

#include <iostream>  
  
using namespace std;  
  
int main()  
{  
 float interest\_rate=12.3;  
 float &fdrate=interest\_rate;// refrence must be initialized with value we   
 //can't leave it empty  
 cout<<"Hello World "<<endl<< fdrate;  
  
 return 0;  
}

**Function overload**

we can’t use same name for each function in c , but in c++ it’s possible the only requirement is we have to differentiate the fucntion by arguments and return types , below example we used absl as function name twice but the parameter and return values are different, so compiler will easily identify the function, it’s called function overloading

#include <iostream>  
  
using namespace std;  
int absl(int a);  
double absl(double a);  
  
int main()  
{  
 int a=-5;  
 double d=-9.0;  
 cout<<absl(a)<<endl;  
 cout<<absl(d)<<endl;  
 return 0;  
}  
int absl(int a)  
{  
 cout<<"From Int"<<endl;  
 return a<0?-a:a;  
}  
double absl(double a)  
{  
 cout<<"From double"<<endl;  
 return a<0?-a:a;  
}

From Int

5

From double

9

**Strings**

In c Language we want to initiate string we have to declar the array , but in c++ we can directly declare like string s1, here string is not a datatype it’s a class , so we creating the object for string i.e s1

we have a multiple way to initialize the string listed below

#include <iostream>  
#include<string.h>  
  
using namespace std;  
  
int main()  
{  
 string s1="c++";  
 cout<<s1<<endl;  
 string s2("Hello");  
 cout<<s2<<endl;  
 string s3=s1;// copy initialization  
 cout<<s3<<endl;  
 string s4(s2);// direct initialization  
 cout<<s4<<endl;  
 string s5(6,'p');// 6 times the character 'p' will concordinate and store to s5  
 cout<<s5<<endl;  
 string s6;  
 // cin>>s6;// if we use space after the word it won't take eg : Hello world, it only take hello  
 //cout<<"You Entered "<<endl<<s6;  
 string s7;  
 getline(cin,s7);  
 cout<<"You Entered "<<endl<<s7;  
   
 return 0;  
}

**String Comparison**

String comparison is based on which string has first highest ascii value.  
#include <iostream>  
#include<string.h>  
  
using namespace std;  
  
int main()  
{  
 string s1="c++";  
 string s2("Hello");  
 if(s1==s2)  
 {  
 cout<<"Equal";  
 }  
 else if(s1<s2)  
 {  
 cout<<"s1 is less than s2 i.e based on ascii value of variable";  
 }else if(s1>s2)  
 {  
 cout<<"s1 is greater i.e based on ascii value of variable";  
 }  
 return 0;  
}

**String Concatenation**

we can directly use + operator to concatenate the strings

#include <iostream>  
#include<string.h>  
  
using namespace std;  
  
int main()  
{  
 string s1="Happy";  
 string s2("Birthday");  
 string s3=s1+" "+s2;  
 // string s4="Happy"+" "+s3; it throw error we can't join two literals( means two strings)  
 string s4="Happy "+s2;// this is possible to concatenate with 1 literal and variable  
 cout<<s4<<endl;  
   
 return 0;  
}

**Range for loop**

#include <iostream>  
#include<string.h>  
  
using namespace std;  
  
int main()  
{  
 string s1="Happy";  
 string s2("Birthday");  
 int integer[]={10,20,30,40,50};  
 for(char c : s1) // for(variableto\_save\_single\_value\_from\_the\_large value : large\_value);  
 {  
 cout<<c<<' ';  
 }  
 cout<<endl;  
 for(int a : integer)  
 {  
 cout<<a<<' ';  
 }  
 return 0;  
}

**ctype.h functions**

isupper(‘A’)--> returns true

isdigit(4)--> returns true

ispunct(@)-->returns true

**Accessing character from string**

#include <iostream>  
#include<string.h>  
  
using namespace std;  
  
int main()  
{  
 string s1="Happy";  
 string s2("Birthday");  
 s1[1]=toupper(s1[2]);// assigning index number 1 with index number 2's content and changing to upper case   
 if(s1.empty())// we can check wheather string is empty or not using this it returns true if string is not empty  
 {   
 cout<<"String is empty";  
 }  
 else  
 {  
 cout<<"String is not empty";  
 }  
 cout<<endl<<s1<<endl;  
 return 0;  
}

**Vectors**

It’s a collection of object of same type,

syntax

vector<data\_type> variable\_name;

eg:

vector<int> v1;

#include <iostream>  
#include<string.h>  
#include<vector>  
using namespace std;  
using std::vector;  
  
  
int main()  
{  
 int a[3]={1,2,3};// Normal c array initialization  
 vector<int> v1; // integer vector initialization syntax : vector<data\_type> variable\_name;  
 vector<string> v2;// string type  
 vector<float> v3;// float type  
 vector<int> v4;// int type  
 vector<int> v5={5,2,2,4,6,1};// we can assign vector variable values using assignment operator  
 vector<int> v6=v5;//we can directly assign one vector contents into another vector if both are same type  
 vector<int> v7(v5);//similary using () we can assign v5 contents will be assigned to v7  
 vector<int> v8{5,8,9,7,5};//without assignment operator we can assign values using ()  
 vector<int> v9(6);//initialze vector with 6 indexing number with default value as 0 , if its char then default null will be assigned  
 vector<int> v10(5,3);//(Numberofelement,initialvalue) in this case 5 index will be created and initialvalue for all will be 3  
 for(auto v: v10)  
 {  
 cout<<v<<' ';  
 }  
 cout<<endl;  
 v8[5]=20;// we try to add additional value into the vector but it won't take it because if the indexing already present and some value is there means we can assign else we can't assign  
 v8[4]=3;// this is possible because index 4 already created and it has value  
 v8.push\_back(6);// if we want to add value at end of the vector we can use push\_back used to append at the end of the vector indexing  
 for(auto i:v8)  
 {  
 cout<<i<<' ';  
 }  
 cout<<endl;  
 cout<<v8[5]<<endl;// printing the appended data here  
 return 0;  
}

**Iterators**

#include <iostream>  
#include<string.h>  
#include<vector>  
using namespace std;  
using std::vector;

int main()  
{  
 vector<int> v={4,5,6,7,8,9,10};  
 auto b=v.begin();// v.begin() fetches the begining address of v and store it to the iterator (it's like pointers)  
 while(b!=v.end())// similarly v.end() fetches end of the address of v we checking b is not a end of the address until run the loop  
 {  
 cout<<\*b<<' ';// to print the value point by the b we need to derefrence it   
 b++;// incrementing the next address  
  
 }  
 return 0;  
}