**Rajesh M**

srimuthurajesh@gmail.com



**C++ Made EASY**

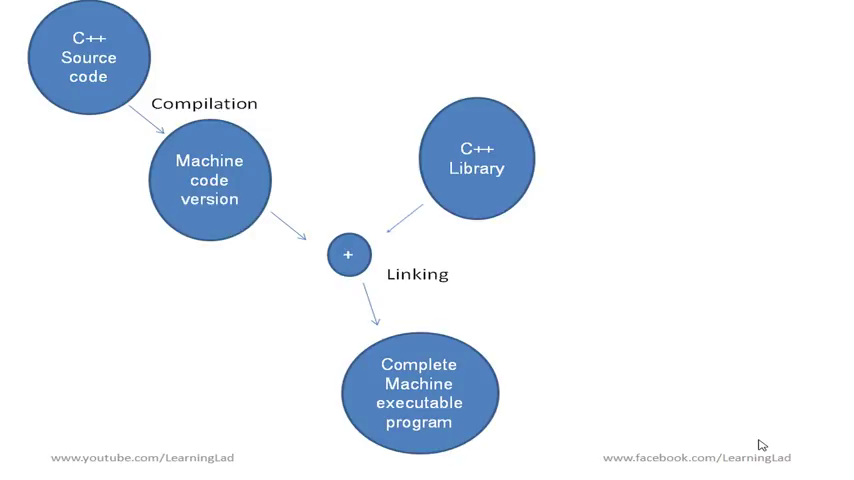
When the world blocks your path by doors of obstacles, don’t just knock it, just sMash it…

Hiii all,

C++ began as expanded version of C, created by Bjarne stroustrup(funny guy still alive ;-) in 1979 at Bell laboratories, NJ. It is initially called as “C with classs”.

Hey oldsport why C++??

Yeah good question, in your journey as a prorgammer, you gonna learn advanced programming languages with lot of functionality like oops etc. for that if u learn c++ it covers all oops concepts with crystal clear, and you can apply it to any advance languages which is object oriented. Don’t worry, c++ is still in use, like mainly Gaming design, visual C++ you can make windows softwares GUI etc. If you thorough in C++ then JAVA is gona damn easy.

*#include <iostream>*

*using namespace std;*

*int main(){*

*}*

**1.Identifiers:** Names that are used to refer variable,fucntion, label and various other user defined objects

**2.Keywords:** All predefined words **48** keywords in C++

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| auto | Double | switch | else | enum |
| break | Extern | float | For | register |
| case | typedef | return | union | goto |
| char | Const | goto | short | unsigned |
| continue | If | signed | sizeof | void |
| int | static | volatile | do | Long |
| struct | While | asm | New | operator |
| template | private | this | protected | throw |
| catch | public | try | Class | friend |
| virtual | Inline | delete |  |  |

**3.Datatype, operators, constant, variables are nearly same to C**

Constant: which is not change during program *const int raj=23;*

Const rajesh obj; //constant object

Void rajesh() const{} void arun() const; //constant function

Constant class only access constant method or variable.

Member intialisationi:

class MyClass {  
public:  
MyClass(int a, int b)  
**: regVar(a), constVar(b)**  
{}  
private:  
int regVar;  
const int constVar;  
};

**Access specifier:**

Public

private

protected-access to derived class

**Class and Object:**

**Object:** Instance of type class

**Class:** Specification of object contains data members and fucntions

Separate file for classes- we need to add rajesh.h rajesh.cpp

|  |  |  |
| --- | --- | --- |
| rajesh | rajesh.h | rajesh.cpp |
| #include <iostream> **#include "rajesh.h"** using namespace std;  int main() { **Rajesh obj;** } | #include “rajesh.cpp”  Class Rajesh { public: Rajesh(); protected: private: }; // this header file is used for variable and constrctors declaration | #include "raejsh.h"  Rajesh::Rajesh() //scope resolution operator { //ctor } |

**Scope resolution operator:** to get the variable, methods from particular class

**Destructor:** used to delete a object ~r (cannot be overload, cant have any parameters)

**The arrow member selection operator(**->**):** whe pointer is using instead of objects

MyClass obj;  
MyClass \*ptr = &obj;  
**ptr->myPrint();**

**Constructor:** 1.Default constructor- Default constructor is the constructor which doesn't take any argument. It has no parameter.

We can intialize it in two methods one normal and another syntax ----classname():variable(45){}

#include <iostream>

using namespace std;

class Cube

{int side; public: Cube(); };

Cube::Cube(){side=89}

int main() {Cube c; cout << c.side; return 0;}

2.Parameterized constructor-

#include <iostream>

using namespace std;

class Cube{

public:

int side;

Cube(int); };

Cube::Cube(int n)

{side=n; }

int main()

{Cube c(5);

cout<<c.side; return 0;}

3.Copy construtor-

#include <iostream.h>

Using namespace std;

Class rectangle{int width, int height;

Public: rectangle();rectangle(int,int); int area(){return (width\*height);}}

rectangle::rectangle(int a, int b){width=a;height=b;}

rectangle::rectangle(){width=5;height=6;}

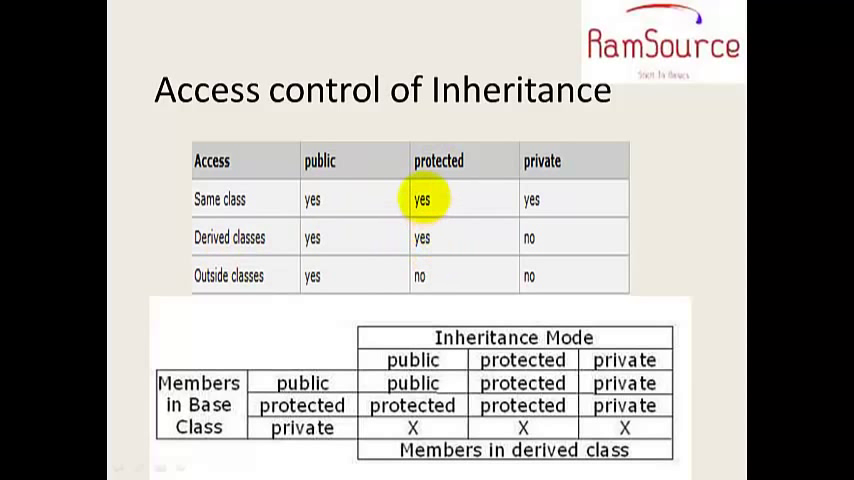
int main() {rectangle obj1(2,4);rectangle obj2; cout<<obj1 area(); cout<<obj2.area();} //Constructor overloading

int main(){rectangle obj1; rectangle onj2=obj1;} // is copy constructor

\*Static member fucntion only can access static member variable

**Fuctions:**

|  |  |
| --- | --- |
| Pass by value | void func(int a, int a, int c){a=+1;b=+1;c=+1;} int main(){int x=1,y=2,z=3; func(x,y,z);} ///passing by value |
| Pass by reference | void func(int &a, int &b, int &c){a=+1;b=+1;c=+1;}  int main(){int x=1,y=2,z=3; func(x,y,z);} //passing to reference |
| Pass by address | void func(int \*p){\*p=3;} int main(){int a=4; func(&a);}  //passing address |
| Default values in parameter | void add(a,b=4){c=a+b;return(c);} main(){add(3,7);} // 7 dan b value, if u not give any value it take 4 |
| Inline function – | same as other normal functions just reduce time only used in short functions not in long functions |
| **Friend funtion:**Public and protected function of class can be access from outside of same class | #include <iostream.h>  Using namesapce std;  Class Box{int width;  friend void printfunc(Box box);//to access object dot operator  Private: void getfunc(double wid);}  Box:: void getfunc(double wid){ wid=width;}  Void printfunc(Box box){ cout<<box.width }//access privte  Int main(){Box box; box.getfunc(83.5); printfunc(box); getch();} |



Optional parameter**:** we give value in function func(int n=5);

Dynamic memory allocation- pointer=new type[i]; delete[]p;

**Inheritance:** Allows us to define a class in terms of another class, instead of writing completely new data members and member functions

#include <iostream>

using namespace std;

class rajesh{

public:

int lenght,width;

int setting(int l, int w){lenght=l;width=w;}};

class rajesh2:public rajesh

{public: int output(){return (lenght\*width);}};

int main()

{rajesh2 m;

m.setting(2,3);

cout<<m.output();

return 0;}

**\*if constructor is beed inherited , construtor of base class first occur and construtor of derived class is execute and it is reverse in case of destrutor**

#include <iostream>

using namespace std;

class rajesh{

public:

rajesh(){cout<<"iam constructor 1"<<endl;}};

class rajesh2:public rajesh

{public: rajesh2(){cout<<"iam construtor 2";}};

int main(){

rajesh2 m;

return 0;}

**POLYMORPHISM:**

**1.Method overriding**

#include <iostream>

using namespace std;

class a{

public:

int x;

virtual int f(){cout<< "hii baby a"<<endl;}};

class b: public a

{public: int f() {cout<<"hii b"<<endl;}};

class c: public b

{public: int f() {cout<<"hii c";}};

int main(){ a ao; b bo; c co;

a\* aptr=&ao; aptr->f();

aptr=&bo; aptr->f();

aptr=&co; aptr->f();}

**2.Operator overloading:**

**1.Uniary without return**

#include <iostream>

using namespace std;

class rajesh{

public:int a,b,c;

rajesh(int A,int B,int C){a=A;b=B;c=C;}

int display(){cout<<a<<b<<c<<endl;}

void operator -(){a=-a;b=-b;c=-c;} };

int main(){

rajesh m(9,5,6);

m.display();

-m;

-m.display();

}

**Virtual function:** two function having same will call virtual first and then other

**Pure virtual function:** is vitual function which has no body. Class contains atleast one pure virtual function is called abstract base class // *virtual int f()=0;*

**Template function:** userdefined datatype.// template<class T>

T fun(T a,T b){

return c+b;

//template<class T> class classname{}

**Encapsultaion:** concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of **data hiding**.

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);

a.addNum(20);

a.addNum(30);

cout << "Total " << a.getTotal() <<endl;

return 0;}

**Data abstraction:** Data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.(representing the implementation)

Data encapsulation is a mechanism of bundling the data, and the functions that use them (wraaping the implementation)

\*Data abstration- show what to show encapsultaion- hide what to not show

\*Data abstraction is achieve through encapsultaion.

**File handling:** <<reading, >>writing, ios::in //open a file for reading ios::out>>open a file for writing

Ios::trunc //if file already exist content will truncated before opening file

Ios::app //append content to end of the file

#include <iostream>

#include <fstream>

using namespace std;

int main(){

char data[100];

ofstream m;

m.open("rajesh.txt");

cout<<"hii baby enter the datas okko"<<endl;

cout<<"enter the name"<<endl;

cin>>data;

m<<data;

m.close();

ifstream n;

n.open("rajesh.txt");

cout<<"hii machi"<<endl;

n>>data;

cout<<data;

n.close();

return 0;}

**Namespace:** when possible to come using two same named function , we use namespace

Namespace first\_space{void fucn(){//info}}

Namespace second\_space{void fucn(){//info}}

Main(){

First\_space::func();

second\_space::func();

}

Or

Namespace first\_space{void fucn(){//info}}

Namespace second\_space{void fucn(){//info}}

Using namespace first\_space;

Main(){

func();

}

**EXCEPTIONS:**

Handles error in the program. Try, catch, throw

*try {int motherAge = 29, sonAge = 36;*

*if (sonAge > motherAge) { throw 99; }}*

*catch (int x) {*

*cout<<"Wrong age values - Error "<<x;*

*} //Outputs "Wrong age values - Error 99"*

Catch(…)// catch all exceptions

**File handling:**

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream MyFileobj; //to open file

MyFileobj.open("test.txt");

MyFileobj << "Some text. \n";

if (MyFileobj.is\_open()) {

MyFileobj << "This is awesome! \n";}

else {cout << "Something went wrong";}

MyFileonj.close(); //close file

}