MY CREATION ,MY DREAM.

" DON'T TRY TO MEMORISE THIS PRPOGRAMS,

JUST TRY TO UNDERSTAND"

------------------------------------------------------------------------------------------------------------------------------------------

Lecture 1:

Introduction to c++:

program:

#include <iostream>

using namespace std;

int main(){

cout<<("Hello world!");

return 0;

}

Lecture 2.1:

Call by value:

program:

#include<iostream>

int call\_by\_value(int a,int b);

using namespace std;

int main()

{

int x,y,r;

cin>>x>>y;

r=call\_by\_value(x,y);

cout<<r;

}

int call\_by\_value(int a,int b)

{ int s;

s=a+b;

return (s);

}

Lecture 2.2:

Call by Reference:

program:

#include<iostream>

int call\_by\_reference(int &a,int &b);

using namespace std;

int main()

{

int x,y,r;

cin>>x>>y;

r=call\_by\_reference(x,y);

cout<<r;

}

int call\_by\_reference(int &a,int &b)

{ int s;

s=a+b;

return (s);

}

Lecture 2.3:

Call by address:

program:

#include<iostream>

int call\_by\_address(int \*a,int \*b);

using namespace std;

int main()

{

int x,y,r;

cin>>x>>y;

r=call\_by\_address(&x,&y);

cout<<r;

}

int call\_by\_address(int \*a,int \*b)

{ int s;

s=\*a+\*b;

return (s);

}

Lecture 3:

Introduction to Function:

#include<iostream>

void fun();

using namespace std;

int main()

{

int x;

cin>>x;

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

fun();

}

void fun()

{

cout<<"i want to be a programmmer";

}

Lecture 4:

Structure :

program:

#include<iostream>

using namespace std;

struct book

{

char title[100];

char author[100];

int bookid;

};

int main()

{

struct book book1;

struct book book2;

strcpy(book1.title,"c program");

strcpy(book2.title,"c++ program");

strcpy(book1.author,"balagurusamy");

strcpy(book2.author,"hervert shill");

book1.bookid=1010;

book2.bookid=1020;

cout<<book1.title<<endl<<book1.author<<endl<<book1.bookid<<endl;

cout<<book2.title<<endl<<book2.author<<endl<<book2.bookid<<endl;}

Lecture 7:

Encapsulation(structure):

#include<iostream>

#include<string>

using namespace std;

struct employ

{

int employid;

char employname[100];

float employsalary;

void input()

{

cout<<"enter employ id ,employ name and salary:";

cin>>employid>>employname>>employsalary;

}

void display()

{

cout<<employid<<employname<<employsalary;

}

};

int main()

{

employ e1;

e1.input();

e1.display();

}

Lecture 5.1:

Class and object:

Program:

#include<iostream>

using namespace std;

class test

{

int a,b;

public:

void set\_value(int x,int y);

void showvalue();

};

void test::set\_value(int x,int y)

{

a=x;b=y;

}

void test::showvalue()

{cout<<a<<b;}

int main()

{

test c1;

c1.set\_value(4,9);

c1.showvalue();

}

Encapsulation:

Lecture 5.2:

Class and object:

program:

#include<iostream>

using namespace std;

class complex

{

int a,b;

public:

void set\_value(int x,int y)

{

a=x;b=y;

}

void showvalue()

{cout<<a<<b;}

};

int main()

{

complex c1;

c1.set\_value(3,4);

c1.showvalue();

}

Lecture 6:

Function call by object and return object:

program:

#include<iostream>

using namespace std;

class complex

{

int a,b;

public:

void set\_data(int x,int y)

{

a=x;b=y;

}

void show\_data()

{

cout<<a<<b<<endl;

}

complex add(complex c)

{

complex temp;

temp.a=a+c.a;

temp.b=b+c.b;

return (temp);

}

};

int main()

{

complex c1,c2,c3;

c1.set\_data(4,7);

c2.set\_data(3,8);

c3=c1.add(c2);

c3.show\_data();

}

Lecture 7.1:

Static member vriable:

program:

#include<iostream>

using namespace std;

class account

{

private:

int balance;

static int roi;

public:

void setBalance(int b)

{

balance=b;

}

void display()

{

cout<<balance<<" "<<roi<<endl;

}

};

int account::roi=6;

int main()

{

account a1,a2;

a1.setBalance(2000);

a2.setBalance(4000);

a1.display();

a2.display();

}

Lecture 7.2:

Static member function:

program:

#include<iostream>

using namespace std;

class account

{

private:

int balance;

static int roi;

public:

void setBalance(int b)

{

balance=b;

}

static void interest(int r)

{

roi=r;

}

void display()

{

cout<<balance<<" "<<roi;

}

};

int account::roi;

int main()

{

account a1;

a1.setBalance(20000);

account::interest(900);

a1.display();

}

Lecture 8.1:

Default Constractor && Parameterised constructor:

program:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

complex()//default constractor

{cout<<"hello constractor"<<endl;}

complex(int x,int y)//parameterized constructor

{a=x;b=y;}

void display()

{

cout<<a<<" "<<b<<endl;

}

};

int main()

{

complex c1,c2(2,3);//initialize at diclaration time(aim of constractor)

c2.display();

}

Lecture 8.2:

Copy constractor:

program:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

complex(int x,int y)//parameterized constractor

{a=x;b=y;}

complex(const complex &c3)//copy constractor

{

a=c3.a;

b=c3.b;}

void display()

{

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

}

};

int main()

{

complex c1(2,3);

//c1.display();

complex c2(c1);

c2.display();}

Lecture 9:

Distructor:

program:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

~complex()

{

cout<<"distractor is working";

}

};

void fun()

{

cout<<"i am an ictian"<<endl;

}

int main()

{

complex c1;

fun();

}

Polymorphism:

Lecture 10.1:

Function overloading:

program:

#include<iostream>

float area(int z);

int area(int a,int b);

using namespace std;

int main()

{

int r;

cout<<"enter the value of redious of circle:";

cin>>r;

float result=area(r);

cout<<"area is "<<result<<endl;

int l,b,ans;

cout<<"enter the value of the length and breath of triangle:";

cin>>l>>b;

ans=area(l,b);

cout<<"area is "<<ans;

}

float area(int z)

{

return (3.1416\*z\*z);

}

int area(int a,int b)

{

return (a\*b);

}

Lecture 10.2:

Function overrinding:

Program:

//early binding hobe,j class er obj sei func execute hobe//

#include<iostream>

using namespace std;

class car

{

public:

int gear;

void setgear(int g)

{gear=g;}

void gearchange()

{ if(gear>3)

gear--;

cout<<"danger∆!decrement your gear";}

void display()

{cout<<gear;}

};

class sportscar:public car

{

public:

void gearchange()

{

if(gear<10)

gear++;

cout<< "increment your gear to win the race"<<endl;}

};

int main()

{

sportscar s;

s.setgear(7);

s.gearchange();

s.display();

}

Lecture 10.3:

Function hiding:

Program:

#include<iostream>

using namespace std;

class car

{

public:

int gear;

void setgear(int hig,int log)

{cout<<"gear difference:"<<hig-log;}

void changegear()

{cout<<"∆∆∆∆∆"<<endl;}

};

class sportscar:public car

{

public:

void setgear(int g)

{cout<<"execute this statement"<<endl;}

};

int main()

{

sportscar s;

s.changegear();

s.setgear(7);

//s.setgear(7,9) it is not possible in c++ but possible in java (function overloading)

}

Lecture 11:

Virtual function:

program:

#include<iostream>

using namespace std;

class car

{

public:

virtual void changegear(int gear)

{ if(gear>3)

gear--;

cout<<"this is a private car,gear is now: "<<gear<<endl;}

};

class sportscar:public car

{

public :

void changegear(int gear)

{if(gear<10)

gear++;

cout<<"this is a sports car,gear incremented.now gear is:"<<gear;}

};

int main()

{car \*cp,c;

sportscar s;

cp=&s;

cp->changegear(7);

}

Operator overloading:

Lecture 12.1 :

Unary operator overloading:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

int setvalue(int x,int y)

{

a=x;b=y;

}

void display()

{ cout<<a<<" "<<b;}

complex operator-()

{ complex temp;

temp.a=-a;

temp.b=-b;

return (temp);

}

};

int main()

{

complex c1,c2;

c1.setvalue(2,4);

c2=c1.operator-();//c2=-c1

c2.display();

}

Lecture 12.2:

Binary operator overloading:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

void setvalue(int x,int y)

{

a=x;b=y;

}

void display()

{

cout<<a<<" "<<b;

}

complex operator+(complex c)

{

complex temp;

temp.a=a+c.a;

temp.b=b+c.b;

return (temp);

}

};

int main()

{

complex c1,c2,c3;

c1.setvalue(3,9);

c2.setvalue(4,7);

c3=c1+c2;//c3=c1.operator+(c2)

c3.display();

}

Lecture 12.2:

Increment Decrement operator overloading:

Program:

#include<iostream>

using namespace std;

class integer

{

private:

int a,b;

public:

void setdata(int x)

{

a=x;

}

integer operator++()//pre increment

{

integer temp;

temp.a=++a;

return (temp);

}

integer operator++(int)//post increment

{

integer temp;

temp.a=a++;

return (temp);

}

void showdata()

{

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

}

};

int main()

{

integer c1,c2;

c1.setdata(5);

c1.showdata();

c2=++c1;

c2=c1++;

c1.showdata();

c2.showdata();

}

Inheritance:

Lecture 13:

Single inheritance:

Program:

Multiple inheritance:

using namespace std;

/\* parent class 1 \*/

class information

{

private:

string name;

int stu\_id;

protected:

void setInfo(){

cin>>name>>stu\_id;

}

void showInfo(){

cout<<endl<<"Stu\_Name: "<<name<<endl<<"Stu\_ID: "<<stu\_id<<endl;

}

};

/\* parent class 2 \*/

class resultMani

{

private:

float marks;

protected:

void setMarks(){

cin>>marks;

}

void processResult(){

if(marks>=80)

cout<<"Marks: "<<marks<<" "<<"Merit/position: Passed"<<endl;

else

cout<<"Marks: "<<marks<<" "<<"Merit/position: Failed"<<endl;

}

};

/\*child class \*/

class student:public information,public resultMani

{

public:

void setData(){

setInfo();

setMarks();

}

void showData(){

showInfo();

processResult();

}

};

int main()

{

student s1,s2;

s1.setData();

s1.showData();

return 0;

}

Program:

Multilevel inheritance :

Program:

#include<iostream>

using namespace std;

class information

{ private: string name;

int st\_id;

protected:

void setInfo()

{ cin>>name>>st\_id; }

public: void showInfo()

{ cout<<endl<<"Stu\_Name: "<<name<<endl<<"Stu\_ID: "<<st\_id<<endl; }

};

class resultMani:public information

{ private: double marks;

protected: void setMarks()

{ cin>>marks; }

void processResult()

{ if(marks>=80)

cout<<"Marks: "<<marks<<" "<<"Merit/position: Passed"<<endl; else cout<<"Marks: "<<marks<<" "<<"Merit/position: Failed"<<endl; } };

class student:public resultMani

{ public: void setData(){ setInfo();

setMarks(); }

void showData()

{ showInfo(); processResult(); } };

int main()

{ student s1,s2; s1.setData(); s1.showData();

return 0; }

Hierchical inheritance:

Program:

#include<iostream>

using namespace std;

class person

{ private: string name,gender;

double age;

protected:

void setData()

{ cin>>name>>gender>>age; }

void showData()

{ cout<<"Name: "<<name<<endl<<"Sex: "<<gender<<" "<<"Age\_Of: "<<age<<endl; } };

class student:public person

{ private: string program;

int studyYear;

public: void setInfo()

{ setData(); // for setting inherit data // cin>>program>>studyYear; }

void showInfo()

{ showData(); // for showing inherit data // cout<<"Program: "<<program<<" "<<"Study\_Year:"<<studyYear<<endl; }

};

class teacher:public person

{ private: string designation ;

int salary;

public: void setInfo()

{ setData(); // for setting inherit data // cin>>designation >>salary; } void showInfo()

{ showData(); // for showing inherit data //

cout<<"Designation : "<<designation <<" "<<"Salary:"<<salary<<endl; } };

int main()

{ student s1,s2;

teacher t1,t2;

s1.setInfo();

s1.showInfo();

t1.setInfo();

t1.showInfo();

return 0; }

Hybrid inheritance:

Program:

Lecture 13.1:

Private inheritance:

Program:

#include<iostream>

using namespace std;

class student

{

protected:

int roll,marks;

};

class result:private student

{

public:

void setData(int x,int y){

roll=x;marks=y; // child class can access the public & protected members of parent class //

}

void showData(){

cout<<"Roll:"<<roll<<" Marks:"<<marks<<endl;

}

};

int main()

{

result obj;

obj.setData(1,50);

obj.showData();

return 0;

Lecture 13.2:

Protected inheritance:

Program:

#include<iostream>

using namespace std;

class student

{

private:

int roll,marks;

protected:

void setValue(){

cin>>roll>>marks;

}

void showResult(){

cout<<"Roll:"<<roll<<" Marks:"<<marks<<endl;

}

};

class result:protected student

{

public:

void setData(){

setValue();

showResult();

}

};

int main()

{

result stu;

stu.setData();

return 0;

}

Lecture 13.3 :

Public inheritance:

#include <iostream>

using namespace std;

class car

{

public:

int gear;

int shiftgear(int g)

{ gear=g;

if(gear<5)

gear++;}

};

class sportscar:public car

{

public:

void display()

{

cout<<"gear is now "<<gear;

}

};

int main()

{ sportscar s1;

s1.shiftgear(3);

s1.display();

}

Lecture 14:

Inline function:

Program:

#include<iostream>

using namespace std;

int summation(int ,int);

int main()

{

int a=5,b=10;

cout<< summation( a,b);

}

inline int summation(int x,int y)

{

int sum;

sum=x+y;

return(sum);

}

Lecture 15:

Friend function:

Program:

Lecture 15.1:

Friend function(2 class use):

#include<iostream>

using namespace std;

class B;

class A

{

private:

int a;

public:

void setdata(int x)

{ a=x;}

friend void fun(A ,B );

};

class B

{

private:

int b;

public:

void setdata(int m)

{b=m;}

friend void fun(A ,B );

};

void fun(A obj3,B obj4)

{ cout<<"sum is:"<<obj3.a+obj4.b;}

int main()

{

A obj1;

B obj2;

obj1.setdata(6);

obj2.setdata(7);

fun(obj1,obj2);}

Lecture 15.2:

Friend function(unary oper. Overloading):

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

void setdata(int x,int y)

{ a=x;b=y;}

void showdata()

{cout<<a<<b;}

friend complex operator-(complex);

};

complex operator-(complex c)

{

complex temp;

temp.a=-c.a;

temp.b=-c.b;

return(temp);}

int main()

{

complex c1,c2;

c1.setdata(5,7);

c2=-c1;

c2.showdata();

}

Lecture 15.3:

Friend function (binary oper overloading):

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

void setdata(int x,int y)

{ a=x;b=y;}

void display()

{cout<<"a="<<a<<"b="<<b;}

friend complex operator+(complex ,complex);

};

complex operator+(complex c4,complex c5)

{

complex c ;

c.a=c4.a+c5.a;

c.b=c4.b+c5.b;

return(c);}

int main()

{ complex c1,c2,c3;

c1.setdata(4,9);

c2.setdata(6,10);

c3=operator+(c1,c2);

c3.display();

}

Pointer:

Lecture 16.1:

Object pointer:

#include<iostream>

using namespace std;

class box

{

private:

int l,w,h;

public:

void setdata(int x,int y,int z)

{l=x;w=y;h=z;}

void showdata()

{

cout<<"l="<<l<<"w="<<w<<"h="<<h;

}

};

int main()

{

box \*p,b1;

p=&b1;

p->setdata(10,5,7);

p->showdata();

}

Lecture 16.2:

This pointer:

#include<iostream>

using namespace std;

class box

{

private:

int l,w,h;

public:

void setdata(int l,int w,int h)

{this->l=l;this->w=w;this->h=h;}

void showdata()

{

cout<<"l="<<l<<"w="<<w<<"h="<<h;

}

};

int main()

{

box b1;

b1.setdata(10,5,7);

b1.showdata();

}

Lecture 17.1:

Type conversion(primitive to class)using constractor:

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

complex()

{};

complex(int k)

{ a=k;b=0;}

void display()

{

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

}

};

int main()

{ complex c1;

int x=5;

c1=x;

c1.display();

}

Lecture 17.2:

Type conversion(class to primitive):

#include<iostream>

using namespace std;

class complex

{

private:

int a,b;

public:

void setvalue(int x,int y)

{

a=x;b=y;

}

void display()

{

cout<<a<<" "<<b<<endl;

}

operator int()

{

return(a);

}

};

int main()

{

complex c1;

c1.setvalue(3,9);

c1.display();

int x;

x=c1;

cout<<x;

}

Lecture 18:

Abstract Class:

program:

#include<iostream>

using namespace std;

class person

{

public:

void flag()

{cout<<"there is no chance to declare, obj. of abstract class"<<endl;}

virtual void fun()=0;//pure virtual fun

};

class student:public person

{

public:

void fun()

{

cout<<"TRY BE A PROGRAMmer"<<endl;}

};

int main()

{

student p1;

p1.fun();

p1.flag();

}

Lecture 19:

Function Template:

program:

#include<iostream>

using namespace std;

int main()

{

cout<<biggest(6,6.5);

cout<<endl;

cout<<biggest(7,6.5);

}

template<class T,class R>

R biggest(T x,R y)

{

if(x>y)

return (x);

else

return(y);

File handling:

Lecture 20.1:

Write in file:

Program:

#include<fstream>

#include<iostream>

using namespace std;

int main()

{

ofstream fout;

fout.open("my file.data");

fout<<"hello";

fout.close();

}

Lecture 20.2:

Read from file:

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ifstream fin;

char ch;

fin.open("My\_File 1.dat");

fin>>ch;

while(!fin.eof()){

cout<<ch;

fin>>ch;

}

return 0;

}

Lecture 20.3:

File using structure:

Program:

#include<bits/stdc++.h>

using namespace std;

struct student

{

char name[200];

int sid;

};

int main()

{

struct student a,b;

strcpy(a.name,"shohag");

a.sid=16030;

ofstream fout;

fout.open("info.txt");

if(!fout)

cout<<"file cant open";

else{

fout.write((char\*)&a,sizeof(struct student));

fout.close();

}

ifstream fin;

fin.open("info.txt");

if(!fout)

cout<<"file cant open";

else{

fin.read((char\*)&b,sizeof(struct student));

f in.close();

cout<<"name:"<<b.name<<" "<<"id:"<<b.sid<<endl;

}

return 0;

}

-----------------------------------------------------------------------------------------------

PERFORMED BY

SHOH@G(ICT-‘13)