**QUEUE:**

#include<iostream>

using namespace std;

const int size=100;

template<class t>

class queue{

private:

t q[size];

int sloc, rloc;

public:

queue()

{

sloc=rloc=0;

}

void qput(t i);

t qget();

};

template<class t>

void queue<t>::qput(t i)

{

if(sloc==size)

{

cout<<"Queue if Full"<<endl;

return;

}

sloc++;

q[sloc]=i;

}

template<class t>

t queue<t>::qget()

{

if(rloc==sloc)

{

cout<<"Queue underflow"<<endl;

return 0;

}

rloc++;

return q[rloc];

}

int main()

{

queue<int>a,b;

a.qput(10);

b.qput(19);

a.qput(20);

b.qput(1);

cout<<a.qget()<<" ";

cout<<b.qget()<<" ";

cout<<a.qget()<<" ";

cout<<b.qget()<<" ";

}

**TIMES:**

#include <iostream>

using namespace std;

template<class X>

void repeat(X data, int times)

{

do{

cout << data << "\n";

times--;

}while(times); }

int main()

{

repeat("This is a test", 3);

repeat(100, 5);

repeat(99.0/2, 4);

return 0;

}

**VECTOR:**

#include<iostream>

using namespace std;

class vector{

private:

int x,y;

public:

vector(): x(0),y(0){};

vector(int xx, int yy): x(xx), y(yy){};

void print();

vector operator + (vector &b);

vector operator - ();

vector operator - (vector &b);

};

void vector::print()

{

cout<<x<<" "<<y<<endl;

}

vector vector::operator + (vector &b)

{

int sum\_x,sum\_y;

sum\_x=x+b.x;

sum\_y=y+b.y;

return vector (sum\_x,sum\_y);

}

vector vector::operator - ()

{

return vector(-x,-y);

}

vector vector::operator - (vector &b)

{

int diff\_x, diff\_y;

diff\_x=x-b.x;

diff\_y=y-b.y;

return vector(diff\_x,diff\_y);

}

int main()

{

vector u(1,1), v(2,2), sum, neg, diff;

sum=u+v;

sum.print();

neg=-sum;

neg.print();

diff=u-v;

diff.print();

}

**LEGS:**

#include<iostream>

using namespace std;

class animal{

protected:

int legs;

public:

//int legs;

virtual void print() const

{

cout<<"Unknown animal type"<<endl;

}

};

class fish: public animal{

public:

fish(int n)

{

legs=n;

}

void print() const

{

cout<<"Fish has "<< legs <<" legs"<<endl;

}

};

class bird: public animal{

public:

bird(int n)

{

legs=n;

}

void print() const

{

cout<<"A bird has "<<legs<<" legs"<<endl;

}

};

class mammal: public animal{

public:

mammal(int n)

{

legs=n;

}

void print() const

{

cout<<"A mammal has "<<legs<<" legs"<<endl;

}

};

int main()

{

animal \*ptr[4];

ptr[0]=new fish(0);

ptr[1]=new bird(2);

ptr[2]=new mammal(4);

ptr[3]=new animal;

for(int i=0; i<4; i++)

ptr[i]->print();

return 0;

}

**CONSTRUCTOR:**

#include<iostream>

using namespace std;

class myclass{

private:

int \*p;

public:

myclass(int i);

myclass(const myclass &obj);

~myclass();

int getvalue()

{

return \*p;

}

};

myclass::myclass(const myclass &obj)

{

p=new int;

\*p=\*obj.p;

cout<<"Copy Constructor Called"<<endl;

}

myclass::myclass(int i)

{

cout<<"Normal Constructor Called"<<endl;

p=new int;

\*p=i;

}

myclass::~myclass()

{

cout<<"Destructor Called"<<endl;

delete p;

}

int main()

{

myclass a(10);

myclass b=a;

return 0;

}

**LAB CONSTRUCTOR:**

#include<iostream>

#include<string.h>

using namespace std;

class emp{

private:

char \*name, \*dept;

double sal, ser;

public:

emp();

emp(char \*nm, char \*dep, double sl, double sr);

void setData();

void setName(char \*nm);

void setDep(char \*dep);

void setSal(double sl);

void setSer(double sr);

void display();

//char& getName();

//char& getDep();

//double getSal();

//double getSer();

emp(const emp &e);

~emp()

{

delete []name;

delete []dept;

cout<<"Destructor Called!"<<endl;

}

};

emp::emp(): sal(0),ser(0)

{

name=new char[20];

strcpy(name,"");

dept=new char[20];

strcpy(dept,"");

}

emp::emp(char \*nm, char \*dep,double sl, double sr): sal(sl), ser(sr)

{

name = new char[strlen(nm)+1];

strcpy(name, nm);

dept = new char[strlen(dep)+1];

strcpy(dept,dep);

}

emp::emp(const emp &e)

{

name=new char[20];

//\*name = e.getName();

dept=new char[20];

//\*dept=e.getDep();

sal=e.sal;

ser=e.ser;

name=e.name;

dept=e.dept;

cout<<"Copy Constructor Called!"<<endl;

}

void emp::setData()

{

cout<<"Enter Name of Employee: ";

cin>>name;

cout<<"Enter Department: ";

cin>>dept;

cout<<"Enter Salary: ";

cin>>sal;

cout<<"Enter Years of Service: ";

cin>>ser;

}

void emp::setName(char \*nm)

{

name = new char[strlen(nm)+1];

strcpy(name, nm);

}

void emp::setDep(char \*dep)

{

dept = new char[strlen(dep)+1];

strcpy(dept,dep);

}

void emp::setSal(double sl)

{

sal=sl;

}

void emp::setSer(double sr)

{

ser=sr;

}

void emp::display()

{

cout<<"Name: "<<name<<endl<<"Department: "<<dept<<endl<<"Salary: "<<sal<<endl<<"Years of Service: "<<ser<<endl;

}

int main()

{

emp e1;

e1.setData();

e1.display();

emp e2(e1);

//e2=e1;

e2.display();

}

#include<iostream>

using namespace std;

class complex{

private:

float real, img;

public:

complex();

complex(float rl,float im);

complex setReal(float rl);

complex setImg(float im);

float getReal();

float getImg();

complex display();

};

complex::complex(): real(0), img(0)

{

}

complex::complex(float rl, float im): real(rl), img(im)

{

}

complex complex::setReal(float rl)

{

real=rl;

return \*this;

}

complex complex::setImg(float im)

{

img=im;

return \*this;

}

complex complex::display()

{

cout<<"("<<real<<" , "<<img<<")"<<endl;

}

int main()

{

float rl,im;

cout<<"Enter Real Part: ";

cin>>rl;

cout<<"Enter Imaginary Part: ";

cin>>im;

complex c1,c2;

c1.setReal(rl).setImg(im).display(); // void functions should be changed to class name

}

**INHERITANCE:**

#include<iostream>

using namespace std;

class person{

protected:

string name,city;

int id,num;

public:

person();

person(string nm, string cty, int idd, int numb);

void setName(string nm);

void setCity(string cty);

void setID(int idd);

void setNum(int numb);

string getName();

string getCity();

int getID();

int getNum();

void display();

};

person::person()

{

}

person::person(string nm, string cty, int idd, int numb)

{

name=nm;

city=cty;

id=idd;

num=numb;

}

void person::setName(string nm)

{

name=nm;

}

void person::setCity(string cty)

{

city=cty;

}

void person::setID(int idd)

{

id=idd;

}

void person::setNum(int numb)

{

num=numb;

}

string person::getName()

{

return name;

}

string person::getCity()

{

return city;

}

int person::getID()

{

return id;

}

int person::getNum()

{

return num;

}

void person::display()

{

cout<<endl<<"ID: "<<id<<endl<<"Name: "<<name<<endl<<"City: "<<city<<endl<<"Contact: "<<num<<endl;

}

class patient: public person{

protected:

string admit;

public:

patient();

patient(int idd, string nm,string cty, int numb, string ad);

patient(string ad, person p);

void setAdmit(string ad);

string getAdmit();

void display();

};

patient::patient()

{

}

patient::patient(int idd, string nm,string cty, int numb, string ad)

{

id=idd;

name=nm;

city=cty;

num=numb;

admit=ad;

}

patient::patient(string ad, person p)

{

admit=ad;

name=p.getName();

city=p.getCity();

id=p.getID();

num=p.getNum();

}

void patient::setAdmit(string ad)

{

admit=ad;

}

string patient::getAdmit()

{

return admit;

}

void patient::display()

{

cout<<endl<<"ID: "<<id<<endl<<"Name: "<<name<<endl<<"City: "<<city<<endl<<"Contact: "<<num<<endl<<"Admit Date: "<<admit<<endl;

}

class outpat: public patient{

private:

string chedate;

public:

outpat();

outpat(string check, patient pt1);

void setchedate(string dis);

string getchedate();

void display();

};

outpat::outpat()

{

name="";

city="";

id=0;

num=0;

admit="";

}

outpat::outpat(string check, patient pt1) // In this class assign all 'get' functions to corresponding variables

{

name=pt1.getName();

city=pt1.getCity();

id=pt1.getID();

num=pt1.getNum();

admit=pt1.getAdmit();

chedate=check;

}

void outpat::setchedate(string check)

{

chedate=check;

}

string outpat::getchedate()

{

return chedate;

}

void outpat::display()

{

cout<<endl<<"ID: "<<id<<endl<<"Name: "<<name<<endl<<"City: "<<city<<endl<<"Contact: "<<num<<endl<<"Admit Date: "<<admit<<endl<<"Checkback Date: "<<chedate<<endl;

}

class respat: public patient{

private:

string disdate;

public:

respat();

respat(string dis, patient pt1);

void setdisdate(string dis);

string getdisdate();

void display();

};

respat::respat()

{

name="";

city="";

id=0;

num=0;

admit="";

}

respat::respat(string dis, patient pt1)

{

name=pt1.getName();

city=pt1.getCity();

id=pt1.getID();

num=pt1.getNum();

admit=pt1.getAdmit();

disdate=dis;

}

void respat::setdisdate(string dis)

{

disdate=dis;

}

string respat::getdisdate()

{

return disdate;

}

void respat::display()

{

cout<<endl<<"ID: "<<id<<endl<<"Name: "<<name<<endl<<"City: "<<city<<endl<<"Contact: "<<num<<endl<<"Admit Date: "<<admit<<endl<<"Discharge Date: "<<disdate<<endl;

}

int main()

{

string nm,cty,ad,check,dis;

int idd, numb;

cout<<"ID: ";

cin>>idd;

cin.ignore();

cout<<"Name: ";

getline(cin,nm);

cout<<"City: ";

getline(cin,cty);

cout<<"Contact: ";

cin>>numb;

cin.ignore();

person p;

p.setName(nm);

p.setCity(cty);

p.setID(idd);

p.setNum(numb);

p.display();

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

cout<<"Enter Admit Date: ";

getline(cin,ad);

patient pt1(idd,nm,cty,numb,ad),pt2(ad,p), pt3;

pt3.setAdmit(ad);

pt1.display();

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

cout<<"Enter Checkback Date: ";

getline(cin,check);

outpat op1(check, pt1), op2;

op2.setchedate(check);

op1.display();

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

cout<<"Enter Discharge Date: ";

getline(cin,dis);

respat rp1(dis, pt1), rp2;

rp2.setdisdate(dis);

rp1.display();

}

**LAB QUEUE**

#include <iostream>

#include <string.h>

using namespace std;

const int MAXSTK = 3;

class book

{

private:

string title[MAXSTK];

int price[MAXSTK], edn[MAXSTK], pages[MAXSTK];

static int TOP;

public:

book()

{

}

void PUSH(string t, int p,int e, int pa)

{

if(TOP == MAXSTK)

{

cout << "STACK is FULL" << endl;

return;

}

else

{

title[TOP] = t;

price[TOP]= p;

edn[TOP]= e;

pages[TOP]= pa;

cout <<t<<" "<< p <<" "<<e<<" "<< pa<< " is stored in location " << TOP << endl;

++TOP;

cout <<"TOP:" <<TOP << endl;

}

}

void POP()

{

if(TOP <= 0)

{

cout << "STACK is EMPTY" << endl;

return;

}

else

{

--TOP;

cout << "ITEM: " << title[TOP] << "\t Loc: " << TOP << endl;

}

}

~book()

{

}

};

int book::TOP = 0;

int main()

{

book b;

b.PUSH("harry potter",100,2,250);

b.PUSH("the hunger games",200,2,350);

b.PUSH("DLD",150,3,200);

b.PUSH("OOP",500,3,300);

cout << endl << endl;

b.POP();

b.POP();

b.POP();

b.POP();

}

#include<iostream>

using namespace std;

const int MAXQ = 3;

class person

{

private:

string fname[MAXQ],lname[MAXQ],add[MAXQ];

int age[MAXQ];

char sex[MAXQ];

static int FRONT;

static int REAR;

public:

person()

{

}

void EnQ(string fn, string ln, int a, char s, string ad)

{

if(REAR == MAXQ && FRONT < 0)

{

cout << "Q is FULL" << endl;

return;

}

else

{

fname[REAR] = fn;

lname[REAR] = ln;

age[REAR] = a;

sex[REAR] = s;

add[REAR] = ad;

cout << "REAR: " << REAR << "\t FRONT: " << FRONT << endl;

cout <<fn<<" "<<ln<<" "<<a<<" "<<s<<" "<<ad << " is stored in location " << REAR << endl;

++REAR;

--FRONT;

}

}

void DeQ()

{

if(FRONT == MAXQ-1 && REAR == 0)

{

cout << "Q is EMPTY" << endl;

return;

}

else

{

++FRONT;

--REAR;

cout << "REAR: " << REAR << "\t FRONT: " << FRONT << endl;

cout << "ITEM: " << fname[FRONT] << "\t Loc: " << FRONT << endl;

}

}

~person()

{

}

};

int person::REAR = 0;

int person::FRONT = MAXQ-1;

int main()

{

person p;

p.EnQ("Usman","Syed",20, 'm',"hayatabad peshawar");

p.EnQ("Riaz","ahmed",22, 'm',"gulbahar peshawar");

p.EnQ("Ali","khan",23, 'm',"hayatabad1 peshawar");

p.EnQ("hamza","rehman",24, 'm',"hayatabad2 peshawar");

cout << endl << endl;

p.DeQ();

p.DeQ();

p.DeQ();

p.DeQ();

p.DeQ();

}

**TRY AND CATCH:**

#include<iostream>

using namespace std;

class compute{

private:

int a,b;

public:

// class for exception

class sum\_of\_ngve {};

compute(int aa, int bb): a(aa), b(bb) {}

void show()

{

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

}

float add(int a, int b)

{

if(a<0 || b<0)

{

throw sum\_of\_ngve();

//return;

}

else

{

float res=a+b;

return res;

}

}

};

int main()

{

int x1=-2,y1=3,z1;

compute c(x1,y1);

try{

cout<<c.add(x1,y1)<<endl;

}

catch(compute::sum\_of\_ngve){

cout<<"Exception: Either of Variable is negative!"<<endl;

}

}