**CECS 282  
EXAMINATION 1 REVIEW**

**The String Class in C++:**

The standard C++ library provides a **string** class type that supports all the operations mentioned above, additionally much more functionality. We will study this class in C++ Standard Library but for now let us check following example:

At this point, you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts.

#include <iostream>

#include <string>

using namespace std;

int main ()

{

string str1 = "Hello";

string str2 = "World";

string str3;

int len ;

// copy str1 into str3

str3 = str1;

cout << "str3 : " << str3 << endl;

// concatenates str1 and str2

str3 = str1 + str2;

cout << "str1 + str2 : " << str3 << endl;

// total lenghth of str3 after concatenation

len = str3.size();

cout << "str3.size() : " << len << endl;

return 0;

}

When the above code is compiled and executed, it produces result something as follows:

str3 : Hello

str1 + str2 : HelloWorld

str3.size() : 10

**Classes**

**Define a class**

#include <iostream>

#include <stdio.h>

using namespace std;

**class** Rectangle {

**int** x, y;

**public**:

**void** set\_values (**int**,**int**);

**int** area (**void**) {**return** (x\*y);}

};

**void** Rectangle::set\_values (**int** a, **int** b) {

x = a;

y = b;

}

**int** main () {

Rectangle rect;

rect.set\_values (3,4);

cout << "area: " << rect.area();

**return** 0;

}

Output

area: 12"

**Define class with a data member and member functions to set and get its value**

**class** MyClass

{

**public**:

**void** setName( string name )

{

name = name;

}

string getName()

{

**return** name;

}

**void** displayMessage()

{

cout << "Welcome to the grade book for " << getName() << "!" << endl;

}

**private**:

string name;

};

**int** main()

{

string n;

MyClass obj;

cout << "Initial name is: " << obj.getName() << endl;

cout << "\nPlease enter the name:" << endl;

getline( cin, n );

obj.setName( n );

cout << endl;

obj.displayMessage();

**return** 0;

}

Output

Initial name is:

Please enter the name:

Joe

Welcome to the grade book for Joe !

**Instantiating multiple objects of the class using its constructor**

**class** MyClass

{

**public**:

MyClass( string name )

{

setName( name );

}

**void** setName( string n )

{

name = n;

}

string getName()

{

**return** name;

}

**void** displayMessage()

{

cout << "Welcome " << getName() << "!" << endl;

}

**private**:

string name;

};

**int** main()

{

MyClass obj1( "A" );

MyClass obj2( "B" );

cout << obj1.getName()<< "\n" << obj2.getName() << endl;

**return** 0;

}

Output

A

B

**Class objects can be assigned to each other using default memberwise assignment**

**class** Date

{

**public**:

Date( **int** = 1, **int** = 1, **int** = 2007 );

**void** print();

**private**:

**int** month;

**int** day;

**int** year;

};

Date::Date( **int** m, **int** d, **int** y )

{

month = m;

day = d;

year = y;

}

**void** Date::print()

{

cout << month << '/' << day << '/' << year;

}

**int** main()

{

Date date1( 8, 8, 2008 );

Date date2;

cout << "date1 = ";

date1.print();

cout << "\ndate2 = ";

date2.print();

date2 = date1;

cout << "\n\nAfter default memberwise assignment, date2 = ";

date2.print();

cout << endl;

**return** 0;

}

date1 = 8/8/2008

date2 = 1/1/2007

After default memberwise assignment, date2 = 8/8/2008

**Inheritance**

**Single Inheritance**

// inheritance using English Distances

#include <iostream>

using namespace std;

enum posneg { pos, neg }; //for sign in DistSign

////////////////////////////////////////////////////////////////

class Distance //English Distance class

{

protected: //NOTE: can't be private

int feet;

float inches;

public: //no-arg constructor

Distance() : feet(0), inches(0.0)

{ } //2-arg constructor)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << "\nEnter feet: "; cin >> feet;

cout << "Enter inches: "; cin >> inches;

}

void showdist() const //display distance

{ cout << feet << "\'-" << inches << '\"'; }

};

////////////////////////////////////////////////////////////////

class DistSign : public Distance //adds sign to Distance

{

private:

posneg sign; //sign is pos or neg

public:

//no-arg constructor

DistSign() : Distance() //call base constructor

{ sign = pos; } //set the sign to +

//2- or 3-arg constructor

DistSign(int ft, float in, posneg sg=pos) :

Distance(ft, in) //call base constructor

{ sign = sg; } //set the sign

void getdist() //get length from user

{

Distance::getdist(); //call base getdist()

char ch; //get sign from user

cout << "Enter sign (+ or -): "; cin >> ch;

sign = (ch=='+') ? pos : neg;

}

void showdist() const //display distance

{

cout << ( (sign==pos) ? "(+)" : "(-)" ); //show sign

Distance::showdist(); //ft and in

}

};

////////////////////////////////////////////////////////////////

int main()

{

DistSign alpha; //no-arg constructor

alpha.getdist(); //get alpha from user

DistSign beta(11, 6.25); //2-arg constructor

DistSign gamma(100, 5.5, neg); //3-arg constructor

//display all distances

cout << "\nalpha = "; alpha.showdist();

cout << "\nbeta = "; beta.showdist();

cout << "\ngamma = "; gamma.showdist();

cout << endl;

return 0;

}

**Overriding functions in the subclasses**

// models employee database using inheritance

#include <iostream>

using namespace std;

const int LEN = 80; //maximum length of names

////////////////////////////////////////////////////////////////

class employee //employee class

{

private:

char name[LEN]; //employee name

unsigned long number; //employee number

public:

void getdata()

{

cout << "\n Enter last name: "; cin >> name;

cout << " Enter number: "; cin >> number;

}

void putdata() const

{

cout << "\n Name: " << name;

cout << "\n Number: " << number;

}

};

////////////////////////////////////////////////////////////////

class manager : public employee //management class

{

private:

char title[LEN]; //"vice-president" etc.

double dues; //golf club dues

public:

void getdata()

{

employee::getdata();

cout << " Enter title: "; cin >> title;

cout << " Enter golf club dues: "; cin >> dues;

}

void putdata() const

{

employee::putdata();

cout << "\n Title: " << title;

cout << "\n Golf club dues: " << dues;

}

};

////////////////////////////////////////////////////////////////

class scientist : public employee //scientist class

{

private:

int pubs; //number of publications

public:

void getdata()

{

employee::getdata();

cout << " Enter number of pubs: "; cin >> pubs;

}

void putdata() const

{

employee::putdata();

cout << "\n Number of publications: " << pubs;

}

};

////////////////////////////////////////////////////////////////

class laborer : public employee //laborer class

{

};

////////////////////////////////////////////////////////////////

int main()

{

manager m1, m2;

scientist s1;

laborer l1;

cout << endl; //get data for several employees

cout << "\nEnter data for manager 1";

m1.getdata();

cout << "\nEnter data for manager 2";

m2.getdata();

cout << "\nEnter data for scientist 1";

s1.getdata();

cout << "\nEnter data for laborer 1";

l1.getdata();

//display data for several employees

cout << "\nData on manager 1";

m1.putdata();

cout << "\nData on manager 2";

m2.putdata();

cout << "\nData on scientist 1";

s1.putdata();

cout << "\nData on laborer 1";

l1.putdata();

cout << endl;

return 0;

}

**Public and private inheritance**  
// tests publicly- and privately-derived classes

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class A //base class

{

private:

int privdataA; //(functions have the same access

protected: //rules as the data shown here)

int protdataA;

public:

int pubdataA;

};

////////////////////////////////////////////////////////////////

class B : public A //publicly-derived class

{

public:

void funct()

{

int a;

a = privdataA; //error: not accessible

a = protdataA; //OK

a = pubdataA; //OK

}

};

////////////////////////////////////////////////////////////////

class C : private A //privately-derived class

{

public:

void funct()

{

int a;

a = privdataA; //error: not accessible

a = protdataA; //OK

a = pubdataA; //OK

}

};

////////////////////////////////////////////////////////////////

int main()

{

int a;

B objB;

a = objB.privdataA; //error: not accessible

a = objB.protdataA; //error: not accessible

a = objB.pubdataA; //OK (A public to B)

C objC;

a = objC.privdataA; //error: not accessible

a = objC.protdataA; //error: not accessible

a = objC.pubdataA; //error: not accessible (A private to C)

return 0;

}

**Levels of inheritance**

// multiple levels of inheritance

#include <iostream>

using namespace std;

const int LEN = 80; //maximum length of names

////////////////////////////////////////////////////////////////

class employee

{

private:

char name[LEN]; //employee name

unsigned long number; //employee number

public:

void getdata()

{

cout << "\n Enter last name: "; cin >> name;

cout << " Enter number: "; cin >> number;

}

void putdata() const

{

cout << "\n Name: " << name;

cout << "\n Number: " << number;

}

};

////////////////////////////////////////////////////////////////

class manager : public employee //manager class

{

private:

char title[LEN]; //"vice-president" etc.

double dues; //golf club dues

public:

void getdata()

{

employee::getdata();

cout << " Enter title: "; cin >> title;

cout << " Enter golf club dues: "; cin >> dues;

}

void putdata() const

{

employee::putdata();

cout << "\n Title: " << title;

cout << "\n Golf club dues: " << dues;

}

};

////////////////////////////////////////////////////////////////

class scientist : public employee //scientist class

{

private:

int pubs; //number of publications

public:

void getdata()

{

employee::getdata();

cout << " Enter number of pubs: "; cin >> pubs;

}

void putdata() const

{

employee::putdata();

cout << "\n Number of publications: " << pubs;

}

};

////////////////////////////////////////////////////////////////

class laborer : public employee //laborer class

{

};

////////////////////////////////////////////////////////////////

class foreman : public laborer //foreman class

{

private:

float quotas; //percent of quotas met successfully

public:

void getdata()

{

laborer::getdata();

cout << " Enter quotas: "; cin >> quotas;

}

void putdata() const

{

laborer::putdata();

cout << "\n Quotas: " << quotas;

}

};

////////////////////////////////////////////////////////////////

int main()

{

laborer l1;

foreman f1;

cout << endl;

cout << "\nEnter data for laborer 1";

l1.getdata();

cout << "\nEnter data for foreman 1";

f1.getdata();

cout << endl;

cout << "\nData on laborer 1";

l1.putdata();

cout << "\nData on foreman 1";

f1.putdata();

cout << endl;

}

**Member functions in multiple inheritance**

// englmult.cpp

// multiple inheritance with English Distances

#include <iostream>

#include <string>

using namespace std;

////////////////////////////////////////////////////////////////

class Type //type of lumber

{

private:

string dimensions;

string grade;

public: //no-arg constructor

Type() : dimensions("N/A"), grade("N/A")

{ }

//2-arg constructor

Type(string di, string gr) : dimensions(di), grade(gr)

{ }

void gettype() //get type from user

{

cout << " Enter nominal dimensions (2x4 etc.): ";

cin >> dimensions;

cout << " Enter grade (rough, const, etc.): ";

cin >> grade;

}

void showtype() const //display type

{

cout << "\n Dimensions: " << dimensions;

cout << "\n Grade: " << grade;

}

};

////////////////////////////////////////////////////////////////

class Distance //English Distance class

{

private:

int feet;

float inches;

public: //no-arg constructor

Distance() : feet(0), inches(0.0)

{ } //constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << " Enter feet: "; cin >> feet;

cout << " Enter inches: "; cin >> inches;

}

void showdist() const //display distance

{ cout << feet << "\'-" << inches << '\"'; }

};

////////////////////////////////////////////////////////////////

class Lumber : public Type, public Distance

{

private:

int quantity; //number of pieces

double price; //price of each piece

public: //constructor (no args)

Lumber() : Type(), Distance(), quantity(0), price(0.0)

{ }

//constructor (6 args)

Lumber( string di, string gr, //args for Type

int ft, float in, //args for Distance

int qu, float prc ) : //args for our data

Type(di, gr), //call Type ctor

Distance(ft, in), //call Distance ctor

quantity(qu), price(prc) //initialize our data

{ }

void getlumber()

{

Type::gettype();

Distance::getdist();

cout << " Enter quantity: "; cin >> quantity;

cout << " Enter price per piece: "; cin >> price;

}

void showlumber() const

{

Type::showtype();

cout << "\n Length: ";

Distance::showdist();

cout << "\n Price for " << quantity

<< " pieces: $" << price \* quantity;

}

};

////////////////////////////////////////////////////////////////

int main()

{

Lumber siding; //constructor (no args)

cout << "\nSiding data:\n";

siding.getlumber(); //get siding from user

//constructor (6 args)

Lumber studs( "2x4", "const", 8, 0.0, 200, 4.45F );

//display lumber data

cout << "\nSiding"; siding.showlumber();

cout << "\nStuds"; studs.showlumber();

cout << endl;

return 0;

}

**Constructors in multiple inheritance**

// multiple inheritance with English Distances

#include <iostream>

#include <string>

using namespace std;

////////////////////////////////////////////////////////////////

class Type //type of lumber

{

private:

string dimensions;

string grade;

public: //no-arg constructor

Type() : dimensions("N/A"), grade("N/A")

{ }

//2-arg constructor

Type(string di, string gr) : dimensions(di), grade(gr)

{ }

void gettype() //get type from user

{

cout << " Enter nominal dimensions (2x4 etc.): ";

cin >> dimensions;

cout << " Enter grade (rough, const, etc.): ";

cin >> grade;

}

void showtype() const //display type

{

cout << "\n Dimensions: " << dimensions;

cout << "\n Grade: " << grade;

}

};

////////////////////////////////////////////////////////////////

class Distance //English Distance class

{

private:

int feet;

float inches;

public: //no-arg constructor

Distance() : feet(0), inches(0.0)

{ } //constructor (two args)

Distance(int ft, float in) : feet(ft), inches(in)

{ }

void getdist() //get length from user

{

cout << " Enter feet: "; cin >> feet;

cout << " Enter inches: "; cin >> inches;

}

void showdist() const //display distance

{ cout << feet << "\'-" << inches << '\"'; }

};

////////////////////////////////////////////////////////////////

class Lumber : public Type, public Distance

{

private:

int quantity; //number of pieces

double price; //price of each piece

public: //constructor (no args)

Lumber() : Type(), Distance(), quantity(0), price(0.0)

{ }

//constructor (6 args)

Lumber( string di, string gr, //args for Type

int ft, float in, //args for Distance

int qu, float prc ) : //args for our data

Type(di, gr), //call Type ctor

Distance(ft, in), //call Distance ctor

quantity(qu), price(prc) //initialize our data

{ }

void getlumber()

{

Type::gettype();

Distance::getdist();

cout << " Enter quantity: "; cin >> quantity;

cout << " Enter price per piece: "; cin >> price;

}

void showlumber() const

{

Type::showtype();

cout << "\n Length: ";

Distance::showdist();

cout << "\n Price for " << quantity

<< " pieces: $" << price \* quantity;

}

};

////////////////////////////////////////////////////////////////

int main()

{

Lumber siding; //constructor (no args)

cout << "\nSiding data:\n";

siding.getlumber(); //get siding from user

//constructor (6 args)

Lumber studs( "2x4", "const", 8, 0.0, 200, 4.45F );

//display lumber data

cout << "\nSiding"; siding.showlumber();

cout << "\nStuds"; studs.showlumber();

cout << endl;

return 0;

}

**Ambiguity in multiple inheritance**

// ambigu.cpp

// demonstrates ambiguity in multiple inheritance

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class A

{

public:

void show() { cout << "Class A\n"; }

};

class B

{

public:

void show() { cout << "Class B\n"; }

};

class C : public A, public B

{

};

////////////////////////////////////////////////////////////////

int main()

{

C objC; //object of class C

// objC.show(); //ambiguous--will not compile

objC.A::show(); //OK

objC.B::show(); //OK

return 0;

}

**Aggregation: Classes within classes**

// containership with employees and degrees

#include <iostream>

#include <string>

using namespace std;

////////////////////////////////////////////////////////////////

class student //educational background

{

private:

string school; //name of school or university

string degree; //highest degree earned

public:

void getedu()

{

cout << " Enter name of school or university: ";

cin >> school;

cout << " Enter highest degree earned \n";

cout << " (Highschool, Bachelor's, Master's, PhD): ";

cin >> degree;

}

void putedu() const

{

cout << "\n School or university: " << school;

cout << "\n Highest degree earned: " << degree;

}

};

////////////////////////////////////////////////////////////////

class employee

{

private:

string name; //employee name

unsigned long number; //employee number

public:

void getdata()

{

cout << "\n Enter last name: "; cin >> name;

cout << " Enter number: "; cin >> number;

}

void putdata() const

{

cout << "\n Name: " << name;

cout << "\n Number: " << number;

}

};

////////////////////////////////////////////////////////////////

class manager //management

{

private:

string title; //"vice-president" etc.

double dues; //golf club dues

employee emp; //object of class employee

student stu; //object of class student

public:

void getdata()

{

emp.getdata();

cout << " Enter title: "; cin >> title;

cout << " Enter golf club dues: "; cin >> dues;

stu.getedu();

}

void putdata() const

{

emp.putdata();

cout << "\n Title: " << title;

cout << "\n Golf club dues: " << dues;

stu.putedu();

}

};

////////////////////////////////////////////////////////////////

class scientist //scientist

{

private:

int pubs; //number of publications

employee emp; //object of class employee

student stu; //object of class student

public:

void getdata()

{

emp.getdata();

cout << " Enter number of pubs: "; cin >> pubs;

stu.getedu();

}

void putdata() const

{

emp.putdata();

cout << "\n Number of publications: " << pubs;

stu.putedu();

}

};

////////////////////////////////////////////////////////////////

class laborer //laborer

{

private:

employee emp; //object of class employee

public:

void getdata()

{ emp.getdata(); }

void putdata() const

{ emp.putdata(); }

};

////////////////////////////////////////////////////////////////

int main()

{

manager m1;

scientist s1, s2;

laborer l1;

cout << endl;

cout << "\nEnter data for manager 1"; //get data for

m1.getdata(); //several employees

cout << "\nEnter data for scientist 1";

s1.getdata();

cout << "\nEnter data for scientist 2";

s2.getdata();

cout << "\nEnter data for laborer 1";

l1.getdata();

cout << "\nData on manager 1"; //display data for

m1.putdata(); //several employees

cout << "\nData on scientist 1";

s1.putdata();

cout << "\nData on scientist 2";

s2.putdata();

cout << "\nData on laborer 1";

l1.putdata();

cout << endl;

return 0;

}

**Virtual functions**

**Normal functions access with pointers (early binding)**

// notvirt.cpp

// normal functions accessed from pointer

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class Base //base class

{

public:

void show() //normal function

{ cout << "Base\n"; }

};

////////////////////////////////////////////////////////////////

class Derv1 : public Base //derived class 1

{

public:

void show()

{ cout << "Derv1\n"; }

};

////////////////////////////////////////////////////////////////

class Derv2 : public Base //derived class 2

{

public:

void show()

{ cout << "Derv2\n"; }

};

////////////////////////////////////////////////////////////////

int main()

{

Derv1 dv1; //object of derived class 1

Derv2 dv2; //object of derived class 2

Base\* ptr; //pointer to base class

ptr = &dv1; //put address of dv1 in pointer

ptr->show(); //execute show()

ptr = &dv2; //put address of dv2 in pointer

ptr->show(); //execute show()

return 0;

}

Output  
Base  
Base

**Virtual functions accessed with pointers (late binding)**

// virt.cpp

// virtual functions accessed from pointer

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class Base //base class

{

public:

virtual void show() //virtual function

{ cout << "Base\n"; }

};

////////////////////////////////////////////////////////////////

class Derv1 : public Base //derived class 1

{

public:

void show()

{ cout << "Derv1\n"; }

};

////////////////////////////////////////////////////////////////

class Derv2 : public Base //derived class 2

{

public:

void show()

{ cout << "Derv2\n"; }

};

////////////////////////////////////////////////////////////////

int main()

{

Derv1 dv1; //object of derived class 1

Derv2 dv2; //object of derived class 2

Base\* ptr; //pointer to base class

ptr = &dv1; //put address of dv1 in pointer

ptr->show(); //execute show()

ptr = &dv2; //put address of dv2 in pointer

ptr->show(); //execute show()

return 0;

}

Output

Derv1  
Derv2

**Abstract class and pure virtual functions**

// virtpure.cpp

// pure virtual function

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class Base //base class

{

public:

virtual void show() = 0; //**pure virtual function**

};

////////////////////////////////////////////////////////////////

class Derv1 : public Base //derived class 1

{

public:

void show()

{ cout << "Derv1\n"; }

};

////////////////////////////////////////////////////////////////

class Derv2 : public Base //derived class 2

{

public:

void show()

{ cout << "Derv2\n"; }

};

////////////////////////////////////////////////////////////////

int main()

{

// Base bad; //can't make object from abstract class

Base\* arr[2]; //array of pointers to base class

Derv1 dv1; //object of derived class 1

Derv2 dv2; //object of derived class 2

arr[0] = &dv1; //put address of dv1 in array

arr[1] = &dv2; //put address of dv2 in array

arr[0]->show(); //execute show() in both objects

arr[1]->show();

return 0;

}

**Virtual functions and Polymorphism**

// virtpers.cpp

// virtual functions with person class

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class person //person class

{

protected:

char name[40];

public:

void getName()

{ cout << " Enter name: "; cin >> name; }

void putName()

{ cout << "Name is: " << name << endl; }

virtual void getData() = 0; //pure virtual func

virtual bool isOutstanding() = 0; //pure virtual func

};

////////////////////////////////////////////////////////////////

class student : public person //student class

{

private:

float gpa; //grade point average

public:

void getData() //get student data from user

{

person::getName();

cout << " Enter student's GPA: "; cin >> gpa;

}

bool isOutstanding()

{ return (gpa > 3.5) ? true : false; }

};

////////////////////////////////////////////////////////////////

class professor : public person //professor class

{

private:

int numPubs; //number of papers published

public:

void getData() //get professor data from user

{

person::getName();

cout << " Enter number of professor's publications: ";

cin >> numPubs;

}

bool isOutstanding()

{ return (numPubs > 100) ? true : false; }

};

////////////////////////////////////////////////////////////////

int main()

{

person\* persPtr[100]; //array of pointers to persons

int n = 0; //number of persons on list

char choice;

do {

cout << "Enter student or professor (s/p): ";

cin >> choice;

if(choice=='s') //put new student

persPtr[n] = new student; // in array

else //put new professor

persPtr[n] = new professor; // in array

persPtr[n++]->getData(); //get data for person

cout << " Enter another (y/n)? "; //do another person?

cin >> choice;

} while( choice=='y' ); //cycle until not 'y'

for(int j=0; j<n; j++) //print names of all

{ //persons, and

persPtr[j]->putName(); //say if outstanding

if( persPtr[j]->isOutstanding() )

cout << " This person is outstanding\n";

}

return 0;

} //end main()

**Virtual base class**

// normbase.cpp

**// ambiguous reference to base class**

class Parent

{

protected:

int basedata;

};

class Child1 : public Parent

{ };

class Child2 : public Parent

{ };

class Grandchild : public Child1, public Child2

{

public:

int getdata()

{ return basedata; } // ERROR: ambiguous

};

// virtbase.cpp

**// virtual base classes**

class Parent

{

protected:

int basedata;

};

class Child1 : virtual public Parent // shares copy of Parent

{ };

class Child2 : virtual public Parent // shares copy of Parent

{ };

class Grandchild : public Child1, public Child2

{

public:

int getdata()

{ return basedata; } // OK: only one copy of Parent

};

**Virtual destructors**

//vertdest.cpp

//tests non-virtual and virtual destructors

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class Base

{

public:

~Base() //non-virtual destructor

// virtual ~Base() //virtual destructor

{ cout << "Base destroyed\n"; }

};

////////////////////////////////////////////////////////////////

class Derv : public Base

{

public:

~Derv()

{ cout << "Derv destroyed\n"; }

};

////////////////////////////////////////////////////////////////

int main()

{

Base\* pBase = new Derv;

delete pBase;

return 0;

}

**typeid operator**

// typeid.cpp

// demonstrates typeid() function

// RTTI must be enabled in compiler

#include <iostream>

#include <typeinfo> //for typeid()

using namespace std;

////////////////////////////////////////////////////////////////

class Base

{

virtual void virtFunc() //needed for typeid

{ }

};

class Derv1 : public Base

{ };

class Derv2 : public Base

{ };

////////////////////////////////////////////////////////////////

void displayName(Base\* pB)

{

cout << "pointer to an object of: "; //display name of class

cout << typeid(\*pB).name() << endl; //pointed to by pB

}

//--------------------------------------------------------------

int main()

{

Base\* pBase = new Derv1;

displayName(pBase); //"pointer to an object of class Derv1"

pBase = new Derv2;

displayName(pBase); //"pointer to an object of class Derv2"

return 0;

}

**Checking the type of a class with dynamic\_cast**

//dyncast1.cpp

//dynamic cast used to test type of object

//RTTI must be enabled in compiler

#include <iostream>

#include <typeinfo> //for dynamic\_cast

using namespace std;

////////////////////////////////////////////////////////////////

class Base

{

virtual void vertFunc() //needed for dynamic cast

{ }

};

class Derv1 : public Base

{ };

class Derv2 : public Base

{ };

////////////////////////////////////////////////////////////////

//checks if pUnknown points to a Derv1

bool isDerv1(Base\* pUnknown) //unknown subclass of Base

{

Derv1\* pDerv1;

if( pDerv1 = dynamic\_cast<Derv1\*>(pUnknown) )

return true;

else

return false;

}

//--------------------------------------------------------------

int main()

{

Derv1\* d1 = new Derv1;

Derv2\* d2 = new Derv2;

if( isDerv1(d1) )

cout << "d1 is a member of the Derv1 class\n";

else

cout << "d1 is not a member of the Derv1 class\n";

if( isDerv1(d2) )

cout << "d2 is a member of the Derv1 class\n";

else

cout << "d2 is not a member of the Derv1 class\n";

return 0;

}

**Changing pointer type with dynamic\_cast**

//dyncast2.cpp

//tests dynamic casts

//RTTI must be enabled in compiler

#include <iostream>

#include <typeinfo> //for dynamic\_cast

using namespace std;

////////////////////////////////////////////////////////////////

class Base

{

protected:

int ba;

public:

Base() : ba(0)

{ }

Base(int b) : ba(b)

{ }

virtual void vertFunc() //needed for dynamic\_cast

{ }

void show()

{ cout << "Base: ba=" << ba << endl; }

};

////////////////////////////////////////////////////////////////

class Derv : public Base

{

private:

int da;

public:

Derv(int b, int d) : da(d)

{ ba = b; }

void show()

{ cout << "Derv: ba=" << ba << ", da=" << da << endl; }

};

////////////////////////////////////////////////////////////////

int main()

{

Base\* pBase = new Base(10); //pointer to Base

Derv\* pDerv = new Derv(21, 22); //pointer to Derv

//derived-to-base: upcast -- points to Base subobject of Derv

pBase = dynamic\_cast<Base\*>(pDerv);

pBase->show(); //"Base: ba=21"

pBase = new Derv(31, 32); //normal

//base-to-derived: downcast -- (pBase must point to a Derv)

pDerv = dynamic\_cast<Derv\*>(pBase);

pDerv->show(); //"Derv: ba=31, da=32"

return 0;

}