**CAR EXAMPLE**

#include <cstring>

using namespace std ;

class Car {

char\* \_brandName ;

int \_numberOfWheels ;

int \_numberOfSeats ;

public :

Car ( const Car& original ) {

cout << "Copy constructor called" ;

\_brandName = nullptr ;

\*this = original ;

}

Car ( ) {

\_numberOfSeats = 0 ;

\_numberOfWheels = 0 ;

\_brandName = nullptr ;

}

Car ( int numberOfWheels ) {

\_numberOfSeats = 0 ;

\_numberOfWheels = numberOfWheels ;

\_brandName = nullptr ;

}

Car ( int numberOfWheels , int numberOfSeats ) {

\_numberOfWheels = numberOfWheels ;

\_numberOfSeats = numberOfSeats ;

\_brandName = nullptr ;

}

Car ( int numberOfWheels , int numberOfSeats , const char\* pName ) {

\_numberOfSeats = numberOfSeats ;

\_numberOfWheels = numberOfWheels ;

\_brandName = new char[ strlen( pName ) + 1 ] ;

strcpy( \_brandName , pName ) ;

}

Car& operator= ( const Car& pCar ) {

if ( &pCar != this ) {

//cout << "Assignment operator called" << endl ;

\_numberOfSeats = pCar.\_numberOfSeats ;

\_numberOfWheels = pCar.\_numberOfWheels ;

delete [] \_brandName ;

\_brandName = new char[ strlen( pCar.\_brandName ) + 1 ] ;

strcpy( \_brandName , pCar.\_brandName ) ;

}

return \*this ;

}

bool operator> ( int pRightParam ) {

return ( \_numberOfWheels > pRightParam ) ;

}

void operator+= ( int pInt ) {

cout << "+= operator called" << endl ;

\_numberOfSeats += pInt ;

}

int getNumberOfSeats ( ) const {

return \_numberOfSeats ;

}

int getNumberOfWheels ( ) const {

return \_numberOfWheels ;

}

const char\* getBrandName ( ) const {

return \_brandName ;

}

void setNumberOfWheels ( int pWheels ) {

\_numberOfWheels = pWheels ;

}

void setNumberOfSeats ( int pSeats ) {

\_numberOfSeats = pSeats ;

}

void setBrandName ( const char\* pName ) {

delete [] \_brandName ;

\_brandName = new char[ strlen( pName ) + 1 ] ;

strcpy( \_brandName , pName ) ;

}

~Car ( ) {

cout << endl << "Destructor is called" << endl ;

delete [] \_brandName ;

\_brandName = nullptr ;

}

} ;

bool operator> ( int pLeft , const Car& pCar ) {

return pLeft > pCar.getNumberOfSeats() ;

}

ostream& operator<< ( ostream& pOstream , const Car& pCar ) {

pOstream << pCar.getBrandName() << " "

<< pCar.getNumberOfSeats() << " "

<< pCar.getNumberOfWheels() << endl ;

return pOstream ;}

#include <iostream>

using namespace std;

#include "Car.h"

void printCarInfo ( Car pCar ) {

pCar.setBrandName("Toyota") ;

cout << pCar.getBrandName() << " - " << pCar.getNumberOfSeats()

<< " - " << pCar.getNumberOfWheels() << endl ;

}

int main ( ) {

Car c(13, 13, "Ford") ;

Car d , e ;

c += 3 ;

cout << "Car info: " << c << "End of info" ;

cout << "Seats: " << c.getNumberOfSeats() << endl ;

cout << "Wheels " << c.getNumberOfWheels() << endl ;

cout << "Make: " << c.getBrandName() << endl ;

d.setBrandName( "Mitsubishi" ) ;

cout << "Wheels: " << c.getNumberOfWheels() << endl ;

cout << "Name: " << c.getBrandName() << endl ;

return 0 ;

}

**BAG – QUIZ**

#include <iostream>

using namespace std ;

class Bag {

int size\_ ;

int\* values\_ ;

void copy\_ ( const int\* pVals , int pSize ) {

if ( pVals != values\_ ) {

delete [] values\_ ;

values\_ = nullptr ;

size\_ = 0 ;

if ( pVals != nullptr ) {

values\_ = new int[ size\_ ] ;

size\_ = pSize ;

for ( int ii = 0 ; ii < size\_ ; ii ++ ) {

values\_[ ii ] = pVals[ ii ] ;

}

}

}

}

public :

Bag ( ) { cout << "$$$BAG" << endl ; }

Bag ( const Bag& pBag ) {

cout << "###BAG" << endl ;

values\_ = nullptr ;

copy\_( pBag.values\_ , pBag.size\_ ) ;

}

Bag ( const int\* pVals , int pSize ) {

cout << "@@@BAG" << endl ;

values\_ = nullptr ;

copy\_( pVals , pSize ) ;

}

bool Equals ( const Bag& pBag ) const {

return ( size\_ == pBag.size\_ && values\_ == pBag.values\_ ) ;

}

bool operator== ( const Bag& pBag ) const {

bool areEquals = false ;

if ( size\_ == pBag.size\_ ) {

for ( int ii = 0 ; ii < size\_ ; ii ++ ) {

areEquals = areEquals && (values\_[ ii ] == pBag.values\_[ ii ]) ;

}

}

return areEquals ;

}

~Bag ( ) { cout << "~~~BAG" << endl ; }

} ;

void areTheSame ( Bag pFirst , Bag pSecond ) {

if ( pFirst.Equals( pSecond )) {

cout << "Are the same" << endl ;

}

}

void areEqual ( const Bag& pFirst , const Bag& pSecond ) {

if ( pFirst == pSecond ) {

cout << "Are equal" << endl ;

}

}

int main ( ) {

int arr[] = { 1, 3, 5 } ;

Bag a, b(arr, 3), c(arr, 3) ;

areTheSame(b,c) ;

areEqual(b,c);

return 0 ;

}

**WALKTHROUGH**

##DC

%%XY

%%XY

%%XY

$$DD

6

7

$$DD

$$DD

6

7

$$DD

$$DD

**OOP Example**

#include <iostream>

using namespace std ;

class Car {

private :

int y ;

public :

Car ( ) { cout << "Car constructor" << endl ; }

Car ( int n ) { y = n ; cout << "Custom constructor" << endl ; }

virtual void Break ( ) = 0 ;

virtual void turnOff ( ) {

cout << "engine off" << endl ;

}

~Car ( ) { cout << "Car destroyed" << endl ; }

} ;

class SUV : public Car {

public :

SUV ( ) : Car( 9 ) {

cout << "SUV constructor" << endl ;

}

void Break ( ) {

cout << "SUV Breaks" << endl ;

}

void turnOff ( ) {

cout << "Get the baby off the backseat" << endl ;

}

~SUV ( ) { cout << "SUV destroyed" ; }

} ;

class Truck : public Car {

public :

void Break ( ) {

cout << "Truck Breaks" << endl ;

}

void soundHorn ( ) const {

cout << "Truck sounds the horn" << endl ;

}

} ;

int main ( ) {

//Car a ;

//a.Break() ;

SUV s ;

s.Break() ;

//Truck t ;

//t.Break() ;

//Car\* carPointer ;

//carPointer = new SUV() ;

//carPointer->Break() ;

//carPointer->soundHorn() ; //COMPILER ERROR - CANNOT BE DONE

}

**TEMPLATE EXAMPLES**

#include <iostream>

using namespace std ;

class Something {

public :

bool operator> ( const Something& pSomething ) {

return true ;

}

} ;

template<typename T>

void swap\_values(T& a, T& b) {

T c;

if ( a > b ) {

c = a;

a = b;

b = c;

}

}

template<>

void swap\_values<Something>(Something& a , Something& b ) {

cout << "We do nothing" << endl ;

}

int main( ) {

double x = 3.14 , y = 0.56 ;

int a = 3 , b = 3 ;

Something s , t ;

swap\_values(a,b);

swap\_values(s,t);

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

return 0 ;

}

**INHERITANCE WITH RESOURCES**

**BetterCar.h**

#ifndef BETTERCAR\_H\_INCLUDED

#define BETTERCAR\_H\_INCLUDED

#include <cstring>

#include <iostream>

#include "Car.h"

using namespace std;

class BetterCar : public Car {

private:

char\* \_class;

public:

BetterCar() : Car() {

\_class = nullptr;

}

BetterCar(const char\* pClass, const char\* pName) : Car(pName) {

\_class = nullptr;

if (pClass != nullptr) {

\_class = new char[strlen(pClass) + 1];

strcpy\_s(\_class, strlen(pClass) + 1, pClass);

}

}

BetterCar& operator= (BetterCar& pCar) {

if (this != &pCar) {

//call the parent's assignment operator.

this->Car::operator=(pCar);

delete[] \_class;

\_class = nullptr;

if (pCar.\_class != nullptr) {

\_class = new char[strlen(pCar.\_class) + 1];

strcpy\_s(\_class, strlen(pCar.\_class) + 1, pCar.\_class);

}

}

return \*this;

}

bool operator== (const BetterCar& pCar) const {

return

this->Car::operator==(pCar) &&

//(\_class == nullptr && pCar.\_class == nullptr) ||

(\_class != nullptr && pCar.\_class != nullptr &&

strcmp(\_class, pCar.\_class) == 0);

}

const char\* carClass() const {

return \_class;

}

};

ostream& operator<< (ostream& pOS, const BetterCar& pCar) {

pOS << (Car)(pCar) << " " << pCar.carClass();

return pOS;

}

#endif

**Car.h**

#ifndef CAR\_H\_INCLUDED

#define CAR\_H\_INCLUDED

#include <cstring>

#include <iostream>

using namespace std;

class Car {

private:

char\* \_name;

public:

Car() { \_name = nullptr; }

Car(const char\* pName) {

\_name = nullptr;

if (pName != nullptr) {

\_name = new char[strlen(pName) + 1];

strcpy\_s(\_name, strlen(pName) +1 , pName);

}

}

Car(const Car& pCar) {

\_name = nullptr;

\*this = pCar;

}

Car& operator= (const Car& pCar) {

if (this != &pCar) {

delete[] \_name;

\_name = nullptr;

if (pCar.\_name != nullptr) {

\_name = new char[strlen(pCar.\_name) + 1];

strcpy\_s(\_name, strlen(pCar.\_name) +1 , pCar.\_name);

}

}

return \*this;

}

bool operator==(const Car& pCar) const {

return //(\_name == nullptr && pCar.\_name == nullptr ) ||

(\_name != nullptr && pCar.\_name != nullptr && strcmp(\_name, pCar.\_name) == 0);

}

const char\* name() const {

return \_name;

}

virtual ~Car() {

delete[] \_name;

\_name = nullptr;

}

};

ostream& operator<< ( ostream& pOS , const Car& pCar ) {

pOS << pCar.name();

return pOS;

}

#endif

**Main.cpp**

#include <iostream>

#include "BetterCar.h"

using namespace std;

int main() {

BetterCar a("Sedan", "GM");

BetterCar b("Sedan" , "GM");

cout << a;

}