

ECE 220 Computer Systems & Programming

Lecture 21 – Overloading, Inheritance & Polymorphism



Constructor

```
class Person{  
    char name[20];  
    int age;
```

```
public:
```

```
    Person(char const *_name, int _age);  
    Person(){};  
    void ShowData();  
};  
Person::Person(char const *_name, int _age){  
    strcpy(name, _name);  
    age = _age;  
}  
int main(){  
    // Person p = {"Alice", 20};  
    Person p("Alice", 20);  
}
```

.....→ Or Person p = Person("Alice", 20);

Operator Overloading

another example: replace function add()

```
#include <iostream>
#include<cmath>

//Example: a class in CPP, which has 2 members: angle and length
class vector{
private:
    double angle;
    double length;
//Constructor
public:
    vector(double a, double l){
        angle=a;
        length=l;
    }
//Default Constructor
vector(){angle=0.0; length=0.0;}

//function scaleVector to scale the length of a vector
void scaleVector(double alpha) {
    length *= alpha;
}

void printVector()
{
    std::cout<<"Vector length is: "<<length<<std::endl;
    std::cout<<"Vector angle is: "<< angle<<std::endl;
}
```

```
vector add(vector b)
{
    vector c;
    double ax=length*cos(angle);
    double bx=b.length*cos(b.angle);
    double ay=length*sin(angle);
    double by=b.length*sin(b.angle);
    c.length=sqrt((ax+bx)*(ax+bx)+(ay+by)*(ay+by));
    c.angle=atan((ay+by)/(ax+bx));
    return c;
}

};
```

```
int main(){
    vector a = {0.50,1};
    vector *d=&a;

    // no access to the private member
    //use C++ methods/functions

    d->scaleVector(5.0);
    d->printVector();
    vector b = {1.0, 2.0};
    vector c = a.add(b);
    c.printVector();
    return 0;
}
```

Operator Overloading

another example: replace function add()

```
int main(){
    vector a = {0.50,1};
    vector *d=&a;

    // no access to the private member
    //use C++ methods/functions

    d->scaleVector(5.0);
    d->printVector();
    vector b = {1.0, 2.0};
    vector c = a.add(b);
    c.printVector();
    return 0;
}
```



```
vector add(vector b)
{
    vector c;
    double ax=length*cos(angle);
    double bx=b.length*cos(b.angle);
    double ay=length*sin(angle);
    double by=b.length*sin(b.angle);
    c.length=sqrt((ax+bx)*(ax+bx)+(ay+by)*(ay+by));
    c.angle=atan((ay+by)/(ax+bx));
    return c;
}

};
```

```
int main(){
    vector a = {0.50,1};
    vector *d=&a;
    //use C++ methods/functions
    d->scaleVector(5.0);
    d->printVector();

    vector b = {1.0, 2.0};
    vector c = a+b;
    c.printVector();
    return 0;
}
```



```
vector operator+(vector b)
{
    vector c;
    double ax=length*cos(angle);
    double bx=b.length*cos(b.angle);
    double ay=length*sin(angle);
    double by=b.length*sin(b.angle);
    c.length=sqrt((ax+bx)*(ax+bx)+(ay+by)*(ay+by));
    c.angle=atan((ay+by)/(ax+bx));
    return c;
}

};
```

Constructor

```
class Person{  
    char name[20];  
    int age;
```

```
public:
```

```
    Person(char const *_name, int _age);  
    Person(){};  
    void ShowData();  
};  
Person::Person(char const *_name, int _age){  
    strcpy(name, _name);  
    age = _age;  
}  
int main(){  
    // Person p = {"Alice", 20};  
    Person p("Alice", 20);  
}
```

.....→ Or Person p = Person("Alice", 20);

Arrays & Pointers & Objects

- Array of objects

```
Person p[2] = {Person("Alice", 20), Person("Bob", 22) };  
p[0].ShowData();  
p[1].ShowData();
```

- Array of pointers to objects

```
Person *ptr[2];  
ptr[0] = new Person("Alice",20);  
ptr[1] = new Person("Bob",22);  
ptr[0]->ShowData();  
ptr[1]->ShowData();
```

- Reference to objects

```
Person &ref = *ptr[0];  
ref.ShowData();
```

```
Person &ref = p[0];  
ref.ShowData();
```

C++

- **Object Oriented Programming (OOP)**

Programming style associated with **class** and **objects** and other concepts like

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Inheritance – Why?

```
class Airplane{
    private:
    int passenger;
    double baggage;
    int crew_man;
    public:
    Airplane(int person, double weight, int crew){
        passenger = person;
        baggage = weight;
        crew_man = crew;
    }
    void Ride(int person){
        passenger += person;
    }
    void Load(double weight){
        baggage += weight;
    }
    void TakeCrew(int crew){
        crew_man += crew;
    }
};
```

```
class Train{
    private:
    int passenger;
    double baggage;
    int length;
    public:
    Train(int person, double weight, int len){
        passenger = person;
        baggage = weight;
        length = len;
    }
    void Ride(int person){
        passenger += person;
    }
    void Load(double weight){
        baggage += weight;
    }
    void AddLength(int len){
        length += len;
    }
};
```

‘Airplane’ and ‘Train’ share many data and functions!

Inheritance

base class

```
class Vehicle{  
private:  
    int passenger;  
    double baggage;  
public:  
    void Ride(int person){passenger += person;}  
    void Load(double weight){baggage += weight;}  
    int getPassenger(){ return passenger;}  
    double getBaggage(){ return baggage;}  
};
```

derived class

```
class Airplane : public Vehicle{  
private:  
    int crew_man;  
public:  
    Airplane(int crew) {crew_man = crew;}  
    void TakeCrew(int crew){crew_man += crew;}  
    int getCrew(){ return crew_man;}  
    void ShowData(){  
        cout<<"<<Airplane>> "<<endl;  
        cout<<"passenger: "<<getPassenger()<<endl;  
        cout<<"baggage: "<<getBaggage()<<endl;  
        cout<<"crew man: "<<getCrew()<<endl;  
    }  
};
```

Airplane class is inherited from Vehicle class.

Inheritance

```
int main(){  
    Airplane a(10);  
    a.ShowData();  
}
```

<<Airplane>>

passenger: 0

baggage: 0

crew man: 10

Vehicle's member

Airplane's member

Airplane object	
	passenger
	baggage
	crew_man

How do we initialize the members from the base class?

compile error!
because the members are private.

```
Airplane(int person, double weight, int crew) {  
    passenger = person;  
    baggage = weight;  
    crew_man = crew;  
}
```

Constructors: Base vs Derived

```
class Base{
public:
    Base(){
        cout<<"Base() called."<<endl;
    }
    Base(int a){
        cout<<"Base(int a) called."<<endl;
    }
};

class Derived: public Base{
public:
    Derived(){
        cout<<"Derived() called."<<endl;
    }
    Derived(int a){
        cout<<"Derived(int a) called."<<endl;
    }
};
```

```
int main(){
    cout<<"<<d1 declared.>>"<<endl;
    Derived d1;
    cout<<"<<d2 declared.>>"<<endl;
    Derived d2(1);
}
```

```
<<d1 declared.>>
Base() called.
Derived() called.
```

```
<<d2 declared.>>
Base() called.
Derived(int a) called.
```

1. The base constructor is called, then the derived one.
2. The default constructor is called for the base class.

How can we call Base(int a)?

Constructors: Base vs Derived

```
class Base{
public:
    Base(){
        cout<<"Base() called."<<endl;
    }
    Base(int a){
        cout<<"Base(int a) called."<<endl;
    }
};

class Derived: public Base{
public:
    Derived(){
        cout<<"Derived() called."<<endl;
    }
    Derived(int a): Base(a){
        cout<<"Derived(int a) called."<<endl;
    }
};
```

```
int main(){
    cout<<"<<d1 declared.>>"<<endl;
    Derived d1;
    cout<<"<<d2 declared.>>"<<endl;
    Derived d2(1);
}
```

```
<<d1 declared.>>
Base() called.
Derived() called.
<<d2 declared.>>
Base(int a) called.
Derived(int a) called.
```

Initializer list

Call the base class constructor that has one integer argument.

1. Initialize Base members:_INITIALIZER List

```
class Vehicle{  
    int passenger;  
    double baggage;  
public:  
    Vehicle(int person, double weight){  
        passenger = person;  
        baggage = weight;  
    }  
    ...  
};
```

```
class Airplane : public Vehicle{  
    int crew_man;  
public:  
    Airplane(int person, double weight, int crew): Vehicle(person, weight) {  
        crew_man = crew;  
    }  
    ...  
};
```

```
int main(){  
    Airplane a(120, 1300.0, 10);  
    a.ShowData();  
}
```

When this constructor is called,
we will first call
Vehicle(int person, double weight).



```
<<Airplane>>  
passenger: 120  
baggage: 1300  
crew man: 10
```

1. Initialize Base members:_INITIALIZER List

```
class Airplane : public Vehicle{  
    int crew_man;  
public:  
    Airplane(int p, double w, int c): Vehicle(p, w) {  
        crew_man = c;  
    }  
    ...  
};
```

```
int x;  
x = 10;
```

```
class Airplane : public Vehicle{  
    int crew_man;  
public:  
    Airplane(int p, double w, int c): Vehicle(p, w), crew_man(c) {  
    }  
    ...  
};
```

```
int x = 10;
```

You can also use_INITIALIZER List for the data member

2. Initialize Base members: Protected Member

```
class Vehicle{  
→ protected:  
    int passenger;  
    double baggage;  
public:  
    Vehicle(){}  
    ...  
};  
  
class Airplane : public Vehicle{  
    int crew_man;  
public:  
    Airplane(int person, double weight, int crew){  
        passenger = person;  
        baggage = weight;  
        crew_man = crew;  
    }  
    ...  
};
```

Access	public members	protected members	private members
Same Class	Y	Y	Y
Derived Class	Y	Y	N
Outside Class	Y	N	N

Protected members of a class A are not accessible outside of A's code, but is accessible from the code of any class derived from A.

Inheritance Another Example:

Git: orthovector_inheritance.cpp

```
#include <iostream>      // used for IO
#include<cmath>

using namespace std;    // creates textual container for
                        // variables and functions

//Example: a class in CPP, which has 2 members: angle and length
class vector{
protected:
    double angle;
    double length;

//Constructor
public:
    vector(double a, double l){
        angle=a;
        length=l;
    }

//Default Constructor
    vector(){angle=0.0; length=0.0;}

//function scaleVector to scale the length of a vector
    void scaleVector(double alpha) {
        length *= alpha;
    }

    void printVector()
    {
        cout<<"Vector length is: "<<length<<endl;
        cout<<"Vector angle is: "<< angle<<endl;
    }
}
```

```
vector operator+(vector b)
{
    vector c;
    double ax=length*cos(angle);
    double bx=b.length*cos(b.angle);
    double ay=length*sin(angle);
    double by=b.length*sin(b.angle);
    c.length=sqrt((ax+bx)*(ax+bx)+(ay+by)*(ay+by));
    c.angle=atan((ay+by)/(ax+bx));
    return c;
}
```

```
class orthovector: public vector
{
protected:
    int d;

public:
    orthovector(int dir, double l) { // dir is 0,1,2,3
                                    // indicating right, up, left, down
        const double halfPI = 1.5707963268;
        d = dir;
        angle = d*halfPI;
        length = l;
    }

    orthovector() { d = 0; angle = 0.0; length = 0.0; }
    double hypotenuse(orthovector b);

};

double orthovector::hypotenuse(orthovector b) {
    if( (d+b.d)%2 == 0 )
    {if (d==b.d)
        return length+b.length;
        else
            return abs(length-b.length);
    }
    return (sqrt( length*length + b.length*b.length ) );
}
```

```

int main()
{
    orthovector c(1,1);
    orthovector d(1,4);
    vector e = c+d; //add returns a vector type
    e.printVector();

    double f = c.hypotenuse(d);
    cout << "f hypotenuse is " << f << endl;
}

```

```

#include <iostream>    // used for IO
#include<cmath>

using namespace std;    // creates textual container for
                        // variables and functions
//Example: a class in CPP, which has 2 members: angle and length
class vector{
protected:
    double angle;
    double length;
//Constructor
public:
    vector(double a, double l){
        angle=a;
        length=l;
    }
//Default Constructor
    vector(){angle=0.0; length=0.0;}

//function scaleVector to scale the length of a vector
    void scaleVector(double alpha) {
        length *= alpha;
    }
}

```

```

vector operator+(vector b)
{
    vector c;
    double ax=length*cos(angle);
    double bx=b.length*cos(b.angle);
    double ay=length*sin(angle);
    double by=b.length*sin(b.angle);
    c.length=sqrt((ax+bx)*(ax+bx)+(ay+by)*(ay+by));
    c.angle=atan((ay+by)/(ax+bx));
    return c;
}

```

```

class orthovector: public vector
{
protected:
    int d;
public:
    orthovector(int dir, double l) { // dir is 0,1,2,3
        // indicating right, up, left, down
        const double halfPI = 1.507963268;
        d = dir;
        angle = d*halfPI;
        length = l;
    }

    orthovector() { d = 0; angle = 0.0; length = 0.0; }
    double hypotenuse(orthovector b);
};

double orthovector::hypotenuse(orthovector b) {
    if( (d+b.d)%2 == 0 )
    {if (d==b.d)
        return length+b.length;
        else
            return abs(length-b.length);
    }
    return (sqrt( length*length + b.length*b.length ) );
}

```

Polymorphism

- A call to a member function will cause a **different function** to be executed depending on the type of the object that invokes the function.
- **Function overriding** allows to have the same function in derived class which is already defined in its base class.

```
class Vehicle{
    public:
    void ShowData(){cout<<"<<Vehicle>> "<<endl;}
};
class Airplane : public Vehicle{
    public:
    void ShowData(){cout<<"<<Airplane>> "<<endl;}
};
class Train : public Vehicle{
    public:
    void ShowData(){cout<<"<<Train>> "<<endl;}
};
```

```
int main(){
    Airplane a(100,300,20);
    a.ShowData();
}
```

<<Airplane>>

*Airplane::ShowData() overrides
Vehicle::ShowData().*

Declared Type vs. Actual Type

```
int main(){
    Airplane a(100,300,20);
    Train t(50,100,30);

    a.ShowData();           <<Airplane>>
    t.ShowData();           <<Train>>

    Vehicle *ptr;
    ptr = &a;
    ptr->ShowData();         <<Vehicle>>

    ptr = &t;
    ptr->ShowData();         <<Vehicle>>

    //ptr->AddLength(10);    Compile Error!
}
```

- Base class pointer (or reference) can point its derived class.
- However, the base class does not have access to its derived class members.

Virtual Function

- Virtual functions are the member function in the base class that is expected to **be redefined in the derived class**.

```
class Vehicle{
public:
    virtual void ShowData(){
        cout<<"<<Vehicle>> "<<endl;
    }
};
class Airplane : public Vehicle{
public:
    void ShowData(){
        cout<<"<<Airplane>> "<<endl;
    }
};
class Train : public Vehicle{
public:
    void ShowData(){
        cout<<"<<Train>> "<<endl;
    }
};
```

```
int main(){
    Airplane a(100,300,20);
    Train t(50,100,30);

    a.ShowData();
    t.ShowData();

    Vehicle *ptr;
    ptr = &a;
    ptr->ShowData();

    ptr = &t;
    ptr->ShowData();
}
```

static binding

dynamic binding

Virtual Function – Why?

```
class City{  
    private:  
        Vehicle *vlist[100];  
        int index;  
    public:  
        City(){ index = 0;}  
        void AddVehicle(Vehicle *v){  
            vlist[index++] = v;  
        }  
        void ShowList(){  
            for(int i=0;i<index;i++)  
                vlist[i]->ShowData();  
        }  
};
```

```
int main(){  
    City Champaign;  
  
    Champaign.AddVehicle(new Airplane(30,100,5));  
    Champaign.AddVehicle(new Train(100,300,10));  
    Champaign.AddVehicle(new Train(130,300,15));  
  
    Champaign.ShowList();  
}
```

Virtual Function – Why?

```
class City{  
private:  
    Vehicle *vlist[100];  
    int index;  
public:  
    City(){ index = 0;}  
    void AddVehicle(Vehicle *v){  
        vlist[index++] = v;  
    }  
    void ShowList(){  
        for(int i=0;i<index;i++)  
            vlist[i]->ShowData();  
    }  
};
```

```
int main(){  
    City Champaign;  
  
    Champaign.AddVehicle(new Airplane(30,100,5));  
    Champaign.AddVehicle(new Train(100,300,10));  
    Champaign.AddVehicle(new Train(130,300,15));  
  
    Champaign.ShowList();  
}
```

We want to print out the full information about **Airplane or Train**.

But, it will only print out **Vehicle**.

We want to manage *base class*, not *derived classes*.

→ Wish to resolve functions at run-time, a.k.a. dynamic binding.

Abstraction – Pure Virtual Function & Abstract Class

- ‘Vehicle’ class will never be instantiated as it is. Instead, it will be either ‘Airplane’ or ‘Train’ object.
- Abstract class cannot be instantiated (pointer is fine) and implemented with one or more “pure” virtual function

```
class Vehicle{  
    public:  
    virtual void ShowData() = 0;  
};  
class Airplane : public Vehicle{  
    public:  
    void ShowData(){  
        cout<<"<<Airplane>> "<<endl;  
    }  
};  
class Train : public Vehicle{  
    public:  
    void ShowData(){  
        cout<<"<<Train>> "<<endl;  
    }  
};
```

<= abstract class (has a pure virtual function)

<= pure virtual function (has no body)

```
int main(){  
    Vehicle *vptr;  
    Vehicle v(100,10);  
}
```

// this is ok
// compile error

*Derived class must define a body for the pure virtual function, otherwise it will also be considered an abstract base class.

Constructor & Destructor

```
class Person{
    char name[20];
    int age;
public:
    Person(char const *_name, int _age){
        strcpy(name, _name);
        age = _age;
        cout<<"constructing name: "<<name<<endl;
    };
    ~Person(){
        cout<<"destroying name: "<<name<<endl;
    };
};
```

```
int main(){
    Person p1 = Person("Alice", 20);
    Person p2 = Person("Bob", 20);
}
```

constructing name: Alice
constructing name: Bob
destroying name: Bob
destroying name: Alice

Constructor & Destructor

```
#include <iostream>
#include <cstring>
using namespace std;

class Person{
    char *name;
    int age;

public:
    Person(char const *_name, int _age);
    Person(){};
    void ShowData(){cout<<name<<" "<<age<<endl;}
    ~Person();
};

Person::Person(char const *_name, int _age){
    name = new char[strlen(_name)+1];
    strcpy(name, _name);
    age = _age;
}

Person::~~Person(){
    cout<<"destructor is working"<<" "<<name<<endl;
    delete []name;
}

int main(){
    // Person p = {"Alice", 20};
    Person p("Alice", 20);
    p.ShowData();
    Person P2("Bob", 30);
    P2.ShowData();
}
```

Alice 20

Bob 30

destructor is working Bob

destructor is working Alice

Copy Constructor

```
class Point{
private:
    int x,y;
public:
    Point(int _x, int _y){x = _x; y = _y;}
    Point(const Point &p){
        x = p.x;
        y = p.y;
        //p.x = 0; // Don't want to allow this
    }
    void ShowData(){ cout<<"("<<x<<" , "<<y<<"><<endl;}
};

int main(){
    Point p1(10,20);
    Point p2(p1);

    p1.ShowData();
    p2.ShowData();
}
```

- Initialize an object using another object (member-by-member).
- If a copy constructor is not provided by the user, it will be automatically inserted (default copy constructor)

Use "const" to prevent modification on p

Shallow Copy

```
class Person{
    private:
        char *name;
        int age;
    public:
        Person(){};
        Person(const char *_name, int _age);
        void ShowData();
        ~Person();
};

Person::Person(const char *_name, int _age){
    name = new char[strlen(_name)+1];
    strcpy(name, _name);
    age = _age;
}

Person::~~Person(){
    delete []name;
}
```

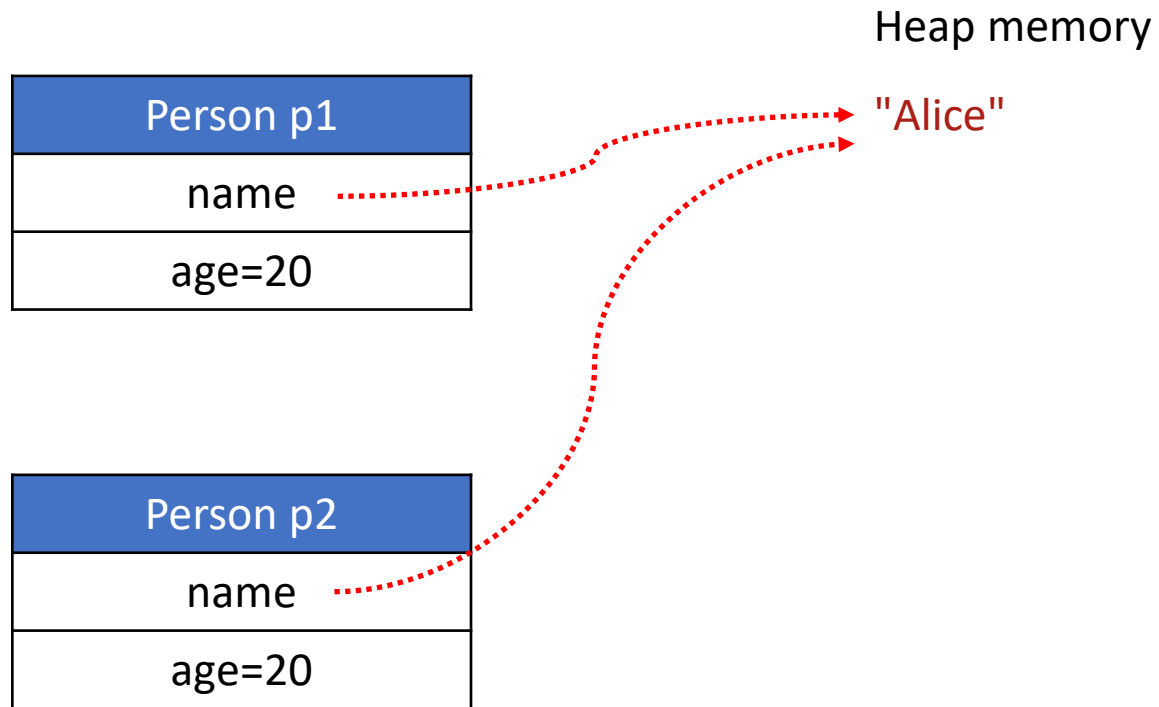
```
int main(){
    Person p1 = Person("Alice", 20);
    Person p2(p1);
    p1.ShowData();
    p2.ShowData();
}
```

Default copy constructor will be inserted.

```
Person::Person(const Person &p){
    name = p.name;
    age = p.age;
}
```

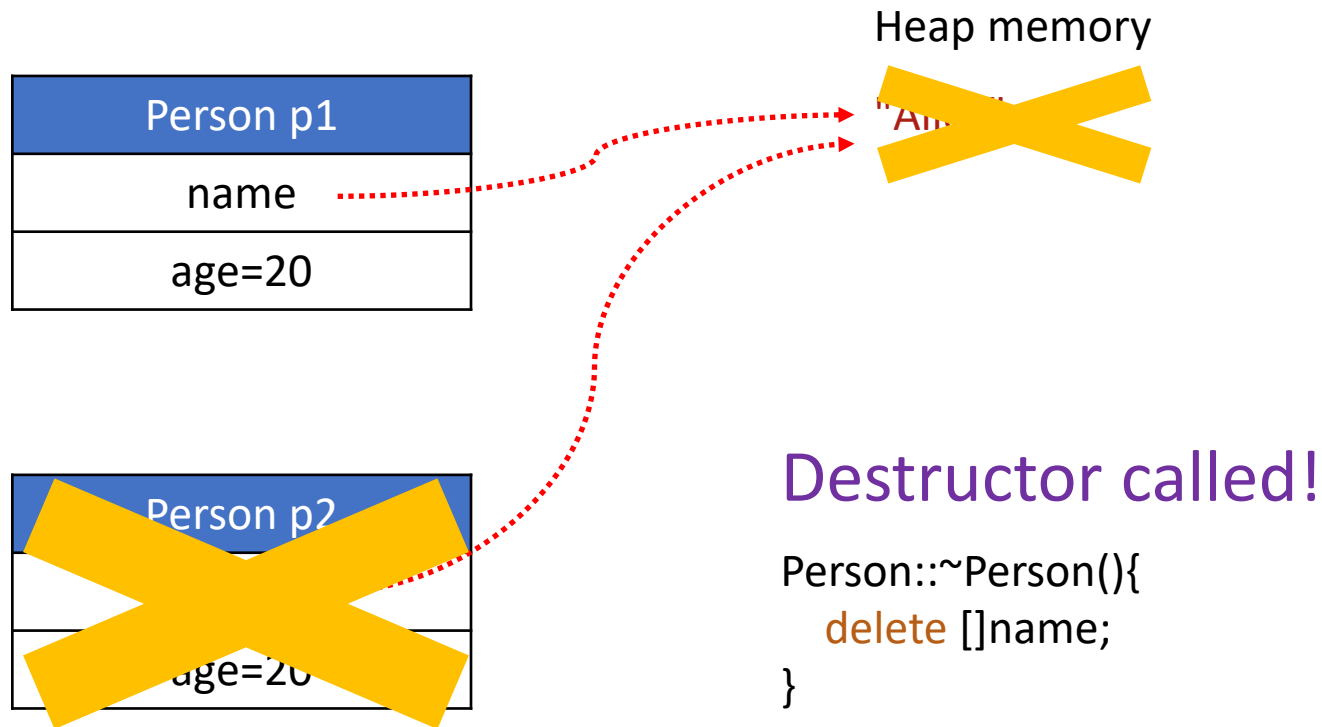
Run-time error!

Shallow Copy



```
Person::Person(const Person &p){  
    name = p.name;  
    age = p.age;  
}
```

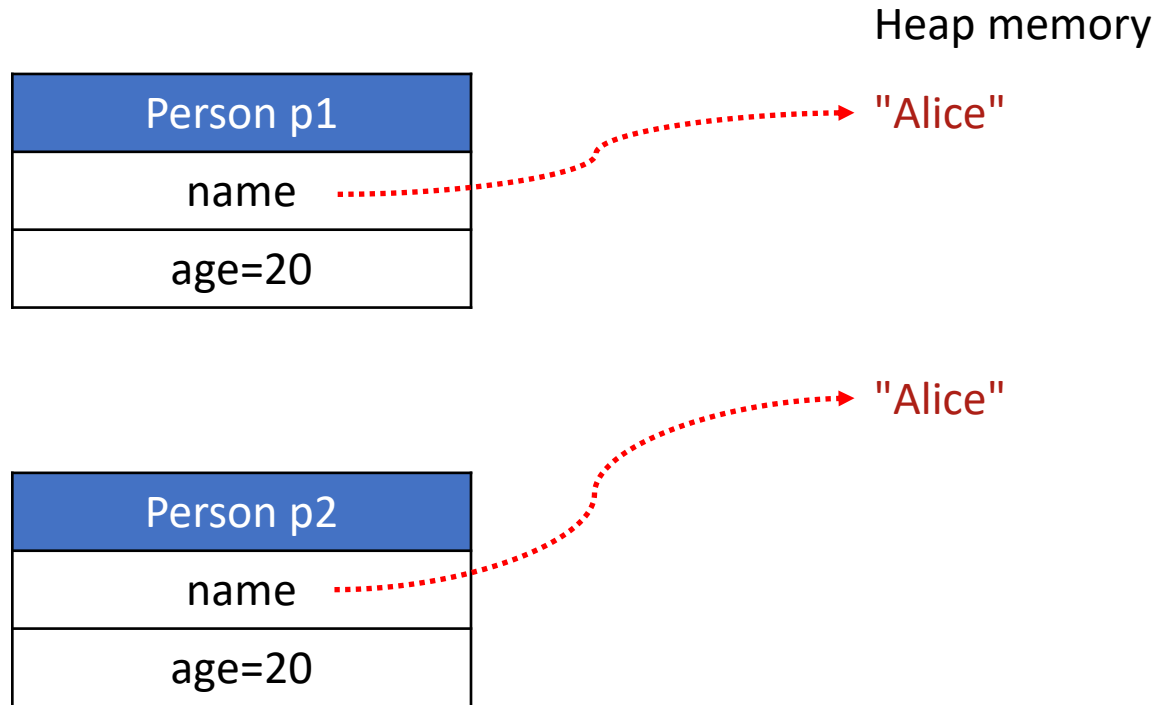
Shallow Copy



When p1 calls its destructor,
the heap memory pointed by “name” is already deallocated.

❌ double free!

Deep Copy

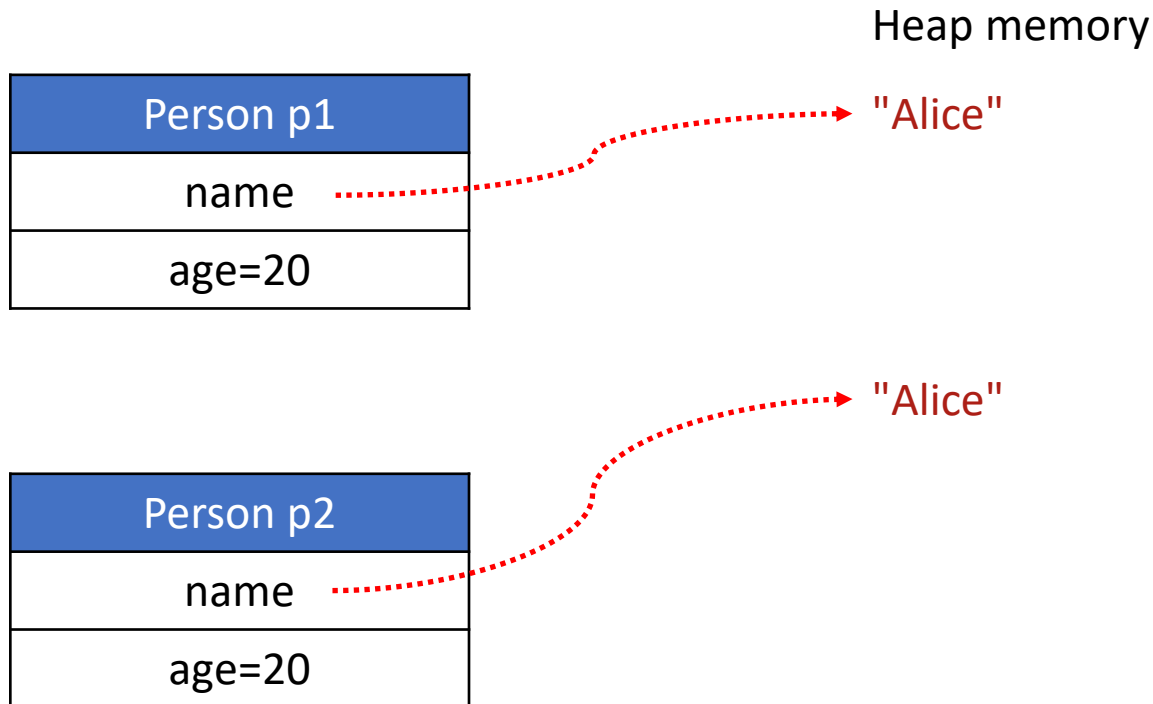


```
Person::Person(const Person &p){  
    name = p.name;  
    age = p.age;  
}
```



```
Person::Person(const Person &p){  
    name = new char[strlen(p.name)+1];  
    strcpy(name, p.name);  
    age = p.age;  
}
```

Deep Copy



```
Person::~~Person(){
    cout<<"destructor is working"<<" " <<name<<endl;
    delete []name;
}

int main(){
    // Person p = {"Alice", 20};
    Person p("Alice", 20);
    p.ShowData();
    Person P2("tom", 30);
    P2.ShowData();
    Person P3(P2);
    P3.ShowData();
}
```

```
#include <iostream>
#include <cstring>
using namespace std;

class Person{
    char *name;
    int age;

public:
    Person(char const *_name, int _age);
    Person();{};
    Person(const Person &p);
    void ShowData(){cout<<name<<" " <<age<<endl;}

    ~Person();
};

Person::Person(char const *_name, int _age){
    name = new char[strlen(_name)+1];
    strcpy(name, _name);
    age = _age;
}

Person::Person(const Person &p){
    name = new char[strlen(p.name)+1];
    strcpy(name, p.name);
    age = p.age;
}
```

Alice 20
tom 30
tom 30
destructor is working tom
destructor is working tom
destructor is working Alice