Object Copying

pm_jat@daiict.ac.in

Consider class A, and a client code using it, and guess what should be output?

```
class A {
public:
    void set(int pos, int x) { data[pos] = x; }
    int get(int pos) { return data[pos]; }
    void print() {
        for (int i=0; i < 5; i++ )
            cout << data[i] << " ";
        cout << endl;
    }
private:
    int data[5];
};
```

```
int main() {
    A a1;
    for (int i=0; i < 5; i++ )
        a1.set(i, i);
    cout << "a1: ";
    a1.print();
    A a2;
    a2 = a1;
    cout << "a2: ";
    a2.print();

    return 0;
}
```

As expected, right ?

```
a1: 0 1 2 3 4
a2: 0 1 2 3 4
```

Now let us add some more stuff in client code …

- What is the output now
- .

  a1: 0 1 2 3 4

  a2: 0 1 2 3 4

  a1: 0 2 4 6 8

  a2: 0 1 2 3 4

- As expected, I guess, you modified data of a1 and that is what it outputs?

```
int main() {
    A a1;
    for (int i=0; i < 5; i++ )
        a1.set(i, i);
    cout << "a1: ";
    a1.print();
    A a2;
    a2 = a1;
    cout << "a2: ";
    a2.print();

    //Let us change data of a1 now
    for (int i=0; i < 5; i++ )
        a1.set(i, i*2);

    cout << "a1: ";
    a1.print();
    cout << "a2: ";
    a2.print();

    return 0;
}
```

Now let us have another implementation of class A, and let us call it as class AD

```
class AD {
public:
    AD() {
        size = 0;
        data = new int[0];
    }
    AD(int sz) {
        size = sz;
        data = new int[sz];
    }
    void set(int pos, int x) { data[pos] = x; }
    int get(int pos) { return data[pos]; }
    void print() {
        for (int i=0; i < size; i++ )
            cout << data[i] << " ";
        cout << endl;
    }
private:
    int *data;
    int size;
};
```

Here is modified client code now
What should be output?

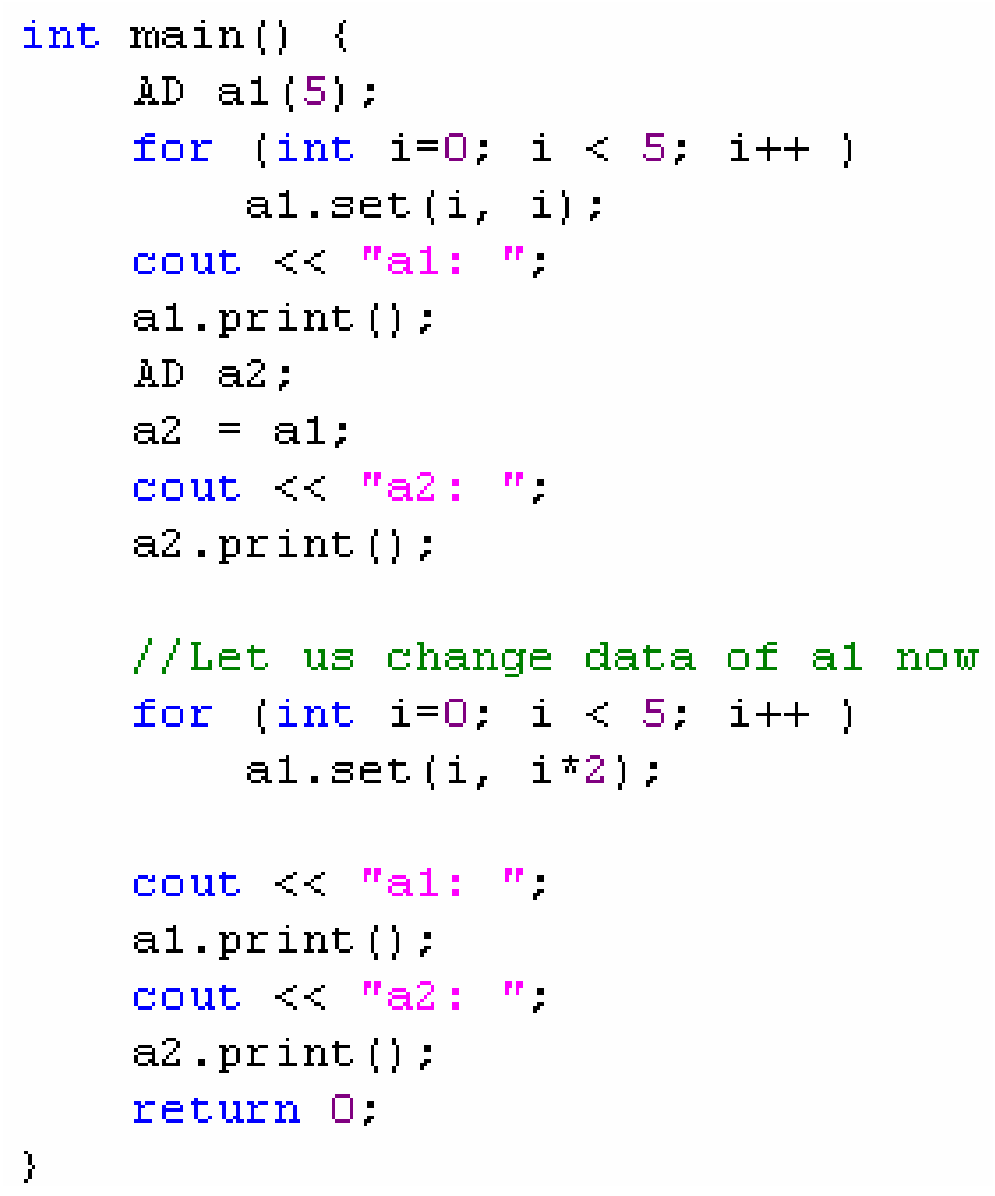
a1: 0 1 2 3 4

a2: 0 1 2 3 4

a1: 0 2 4 6 8

a2: 0 2 4 6 8

- Probably, we did not mean this?



```
int main() {
    AD a1(5);
    for (int i=0; i < 5; i++ )
        a1.set(i, i);
    cout << "a1: ";
    a1.print();
    AD a2;
    a2 = a1;
    cout << "a2: ";
    a2.print();

    //Let us change data of a1 now
    for (int i=0; i < 5; i++ )
        a1.set(i, i*2);

    cout << "a1: ";
    a1.print();
    cout << "a2: ";
    a2.print();
    return 0;
}
```

What does really it mean …

- Recall that each object has its own storage;
- In C++, by default, whenever needs to be copied, it is done bitwise
- In assignments here, it would do bit-wise copy of object a1 (its data) object to a2 (to its data)
- It was OK for class A, but not OK for class AD ?

Shallow copy and deep copy

- What actually went wrong in case of class AD is, while doing bit wise copying, address of external storage in source object is copied to target object;
- as a result a1 and a2 are sharing some storage, that should not happen; each object should have exclusive storage
- A concern you need to take care, for classes which you define !!

Shallow copy and deep copy

- What really we mean by deep copy is, if there is any address (pointer) as data member, copy should not copy that pointer, actually it should copy data to which the pointer points, in the address space of target object.
- This issue, sometimes could be at multiple depth levels, same needs to be applied at all levels.

- Some more examples

Should copying is issue here ?

- You create two employee objects as following

```
Employee e1("Amit", 30000), e2;
```

- And then assign, as follows

```
e2 = e1;
```

- Given definition of Employee class next ..

Employee class definition

```
#include <string>
using namespace std;

class Employee {
public:
    //Constructors
    Employee();
    Employee(string employee_name, double initial_salary);
    //Other Functions
    void setSalary(double new_salary);
    double getSalary() const;
    string getName() const;
private:
   string name;
   double salary;
};
```

Employee class Implementation

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

#include "Employee.h"

Employee::Employee()
{
   name = "No Name";
   salary = 0;
}

Employee::Employee(string employee_name, double initial_salary)
{
   name = employee_name;
   salary = initial_salary;
}

void Employee::setSalary(double new_salary)
{
   salary = new_salary;
}
```

```
double Employee::getSalary() const
{
   return salary;
}

string Employee::getName() const
{
   return name;
}
```

Consider Student Class ver 1.1

```
class Student {
public:
        Student();
        Student(char* name, float
     .

private:
        const long ID;
        char* name;
        float cpi;
        static long next_id;
};
```

```
long Student::next_id = 101;

Student::Student() : ID ( next_id++ )
{
    this->name = new char[8];
    strcpy(this->name, "No Name");
    this->cpi = 0;
}

Student::Student(char* name, float cpi) : ID ( next_id++ )
{
    this->name = new char[strlen(name)+1];
    strcpy(this->name, name);
    this->cpi = cpi;
}
```

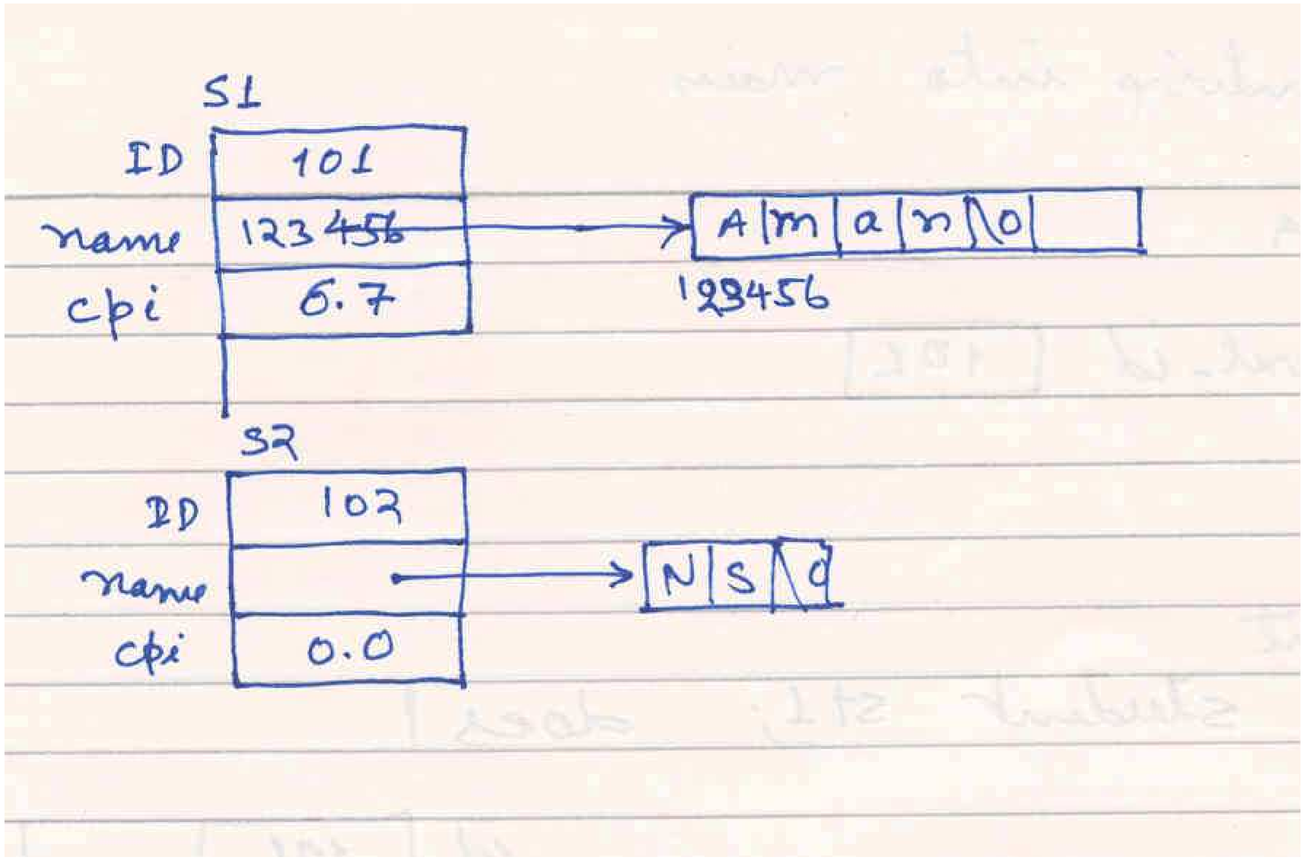
Should copying is issue in this case?

```
Student s1("Aman", 6.7), s2;
s2 = s1;
s2.showdetails()
```

- Considering default copy behavior of C++, let us see what happens

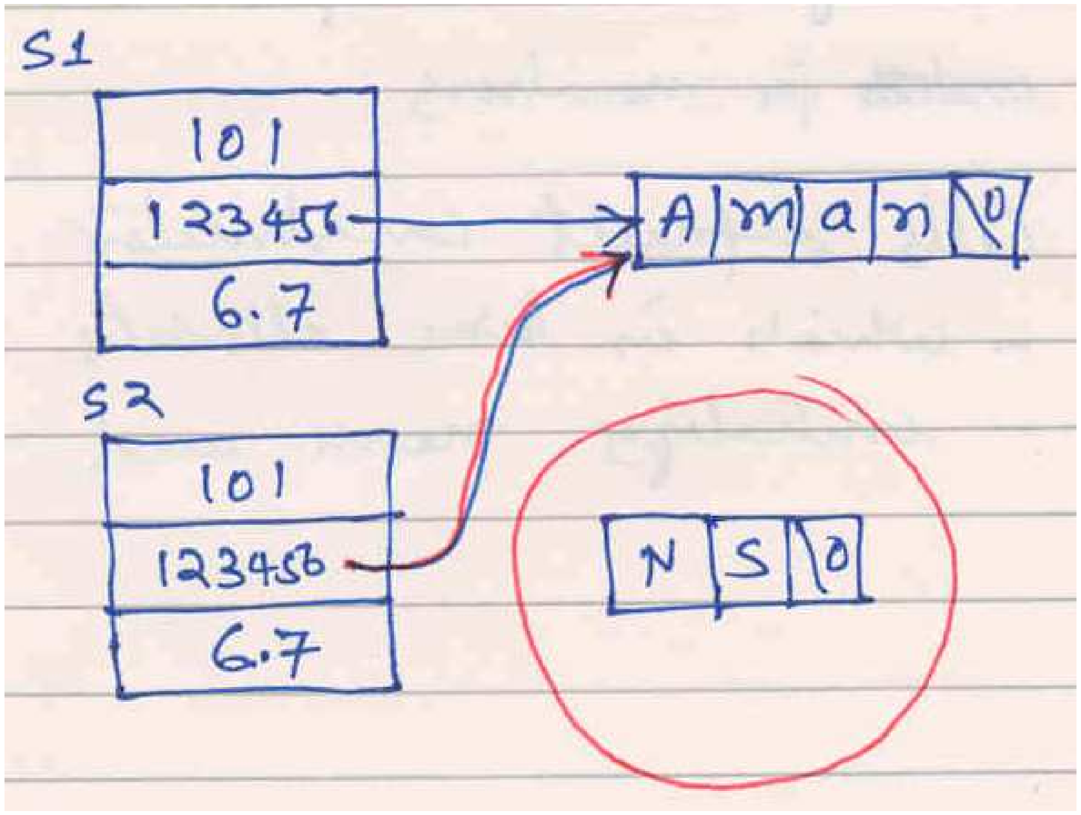
Object definition creates two objects, as shown below

```
Student s1("Aman", 6.7), s2;
```



It is shallow copying

```
s2 = s1;
```



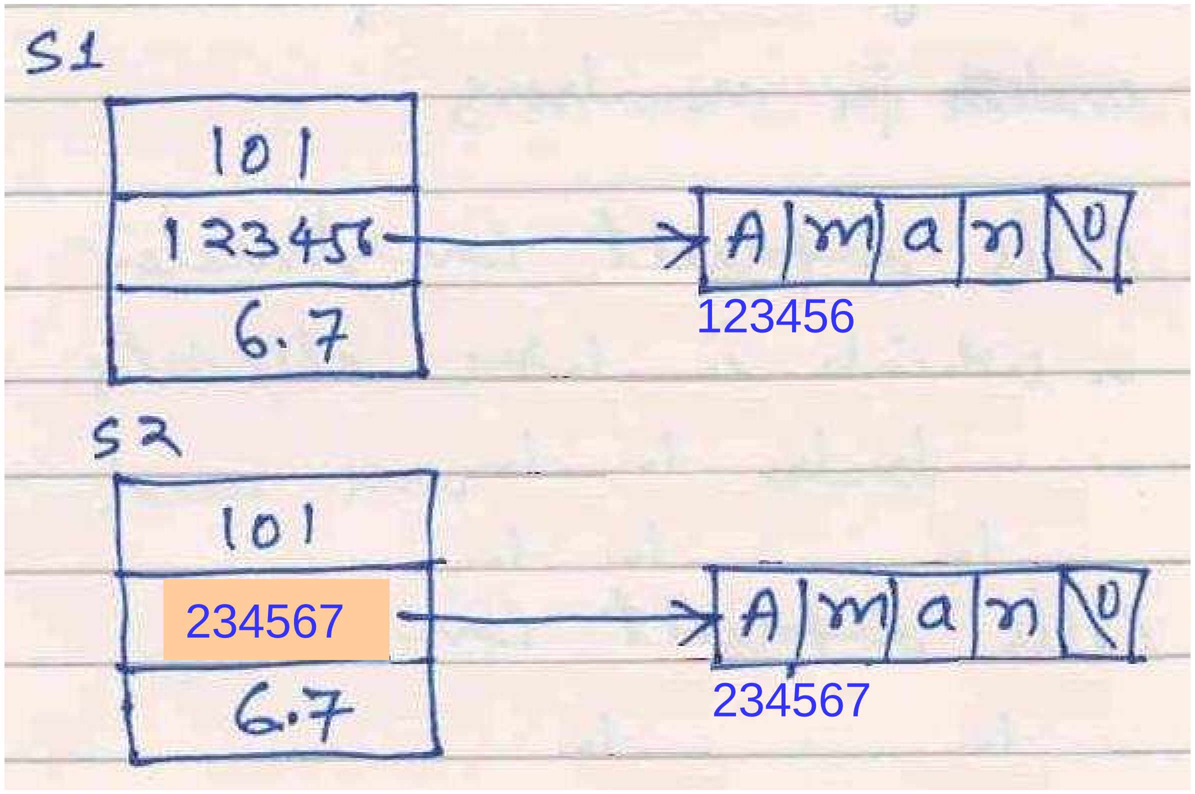
- Both object share some storage, that is not correct; if you modify name of student s1, name of s2 is also modified, and vice-versa

Side effect of shallow copy?

```
Student s1("Aman", 6.7), s2;
s2 = s1;
s1.setName("XYX");
s2.showdetails()
```

- Name of s2 will also set to XYZ

This is what should have been done .. Deep copying



- Both object have separate storage, copies are independent, modification of one does not affect another

- How do we help it ?
- Customize (define) *copying* operation for the class

Typically you copy, when

- Copying an existing object to another existing object, that is assignment
- Copying at the time of object construction, that is initialization
- Note that here we are talking about objects of same type, if their types are different some conversion issues come to picture.

You may have to define these copying operation for a class

- When default behavior of copying (that is bit by bit copying) does not work for a class you need to define copying operation for that class.

- In C++,
  – assignment operation is defined by *overloading assignment operator*
  – Initialization operation (initialing with another existing object of same class) is defined by defining *copy constructor*

Copy Constructor

- Now, let use see how do we define copy constructor.

- Here, we must be able to answer following questions
  - What is a copy constructor ?
  - When do we need to define a copy constructor?
  - How do we do it?
  - How copy constructor is called?
  - What happens when you do not define a copy constructor

What is a copy constructor ?

- Is a constructor, which is used to construct an object, when you attempt to initialize it with some existing object of same class

When do we need to define a copy constructor?

- Compulsorily, when you need to have deep copying

- There could be other reasons, when you want to customize, construction while initializing with object of same type, for example when s2 is being initialized with s1, but you do want ID of s2 to be auto-generated !

How do we define Copy Constructor

- Define a constructor which takes reference to same type of the class
- For example

```
class Student {
public:
      Student();
      Student(char* name, float cpi);
      Student(Student&);
      ~Student();
...
};
```

```
Student::Student(Student& from ) : ID ( next_id++ )
{
    this->name = new char[strlen(from.name)+1];
    strcpy(this->name, from.name);
    this->cpi = from.cpi;
}
```

How copy constructor is called?

- Copy constructor is invoked whenever a new copy of an object is needed to be made
- Typically it happens, when
  - you initialize a object with another existing object
  - You pass a parameter by value
- For example

```
Student s1("Aman", 7.5), s2 = s1;
```

- Or, when

```
process(Student s)
{
  ...
}
```

What happens when you do not define a copy constructor ?

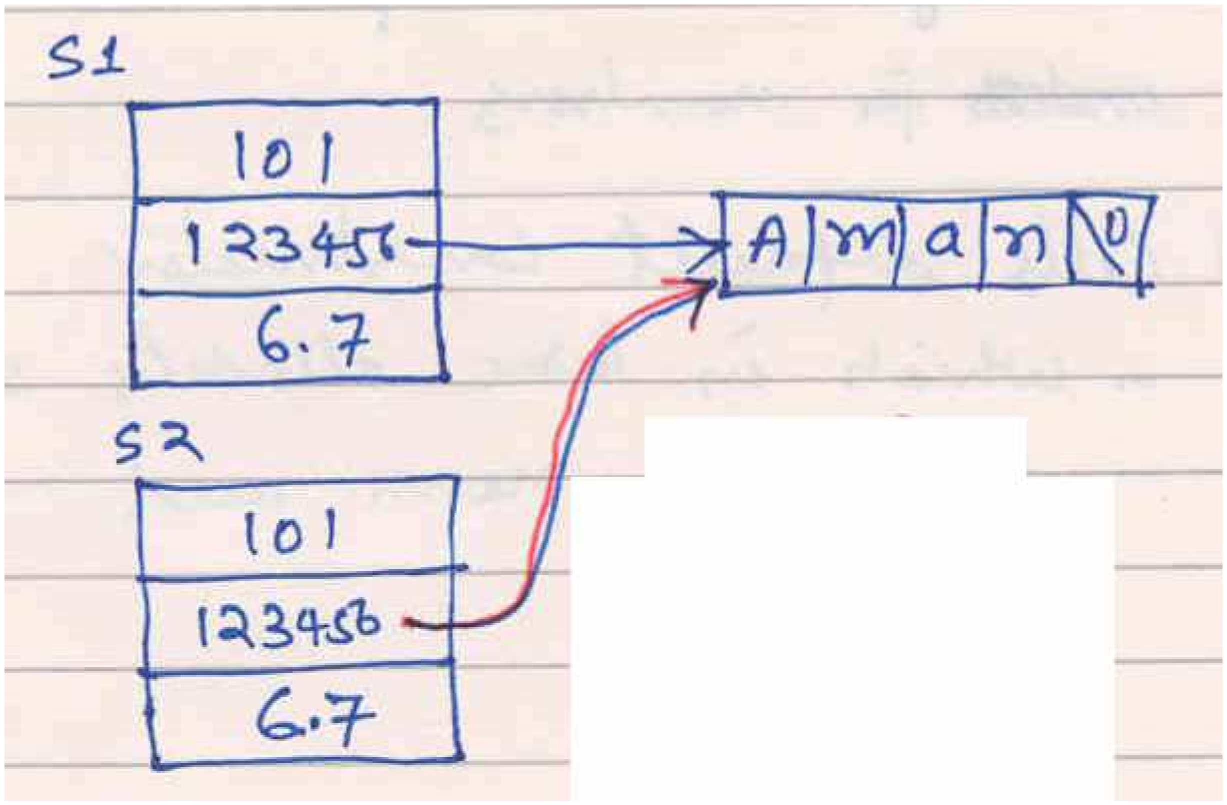
- Compiler provides one, and has a default copying behavior, that is bit by bit copying

- In our student class, when you do not have copy constructor defined, and write following statement-

```
Student s1("Aman", 7.5), s2 = s1;
```

- What happens?

This is what happens



- Again shallow copying

Copy Constructor: we have const reference parameter to same class

```
class Student {
public:
      Student();
      Student(char* name, float cpi);
      Student(const Student&);
      ~Student();
...
};
```

```
Student::Student(const Student& from ) : ID ( next_id++ )
{
    this->name = new char[strlen(from.name)+1];
    strcpy(this->name, from.name);
    this->cpi = from.cpi;
}
```

Exercise

- Considering definition of Copy constructor of student class given here, what should be output of following code-

```
void process(Student s) {
  s.showDetails();
}
int main() {
  Student s("ABC",7.5);
  process( s );
  return 0;
}
```

<u>Thanks</u>