IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: More on Inheritance

Recap

IIT Bombay

- Inheritance with public, private and protected members
- Public, private and protected inheritance/derivation
- Access control in derived classes

Overview of This Lecture

IIT Bombay

- Redefining member functions of the base class
- Access methods of base class using derived classes
- Constructors for derived classes
- Destructors
- Inheritance of assignment operators

Acknowledgment

- Much of this lecture is motivated by the treatment in

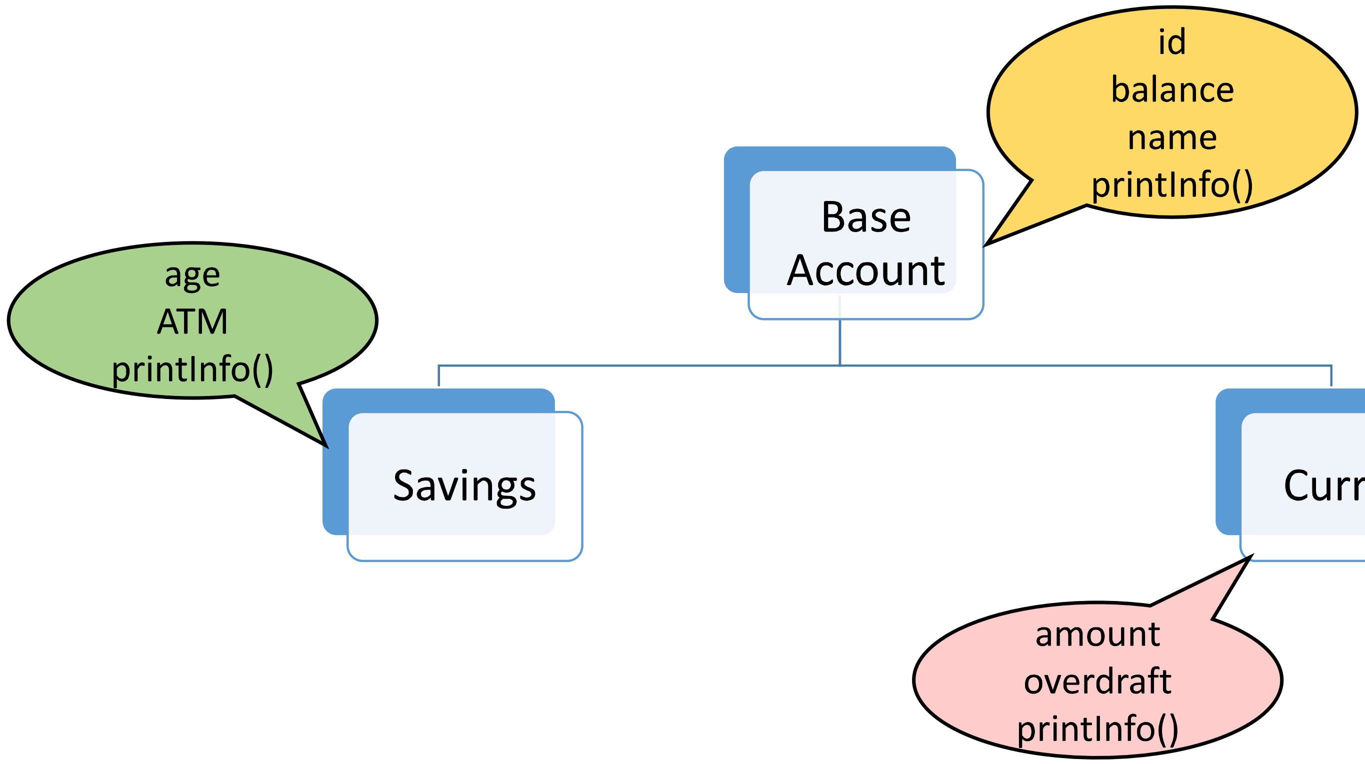
  An Introduction to Programming Through C++

  by Abhiram G. Ranade

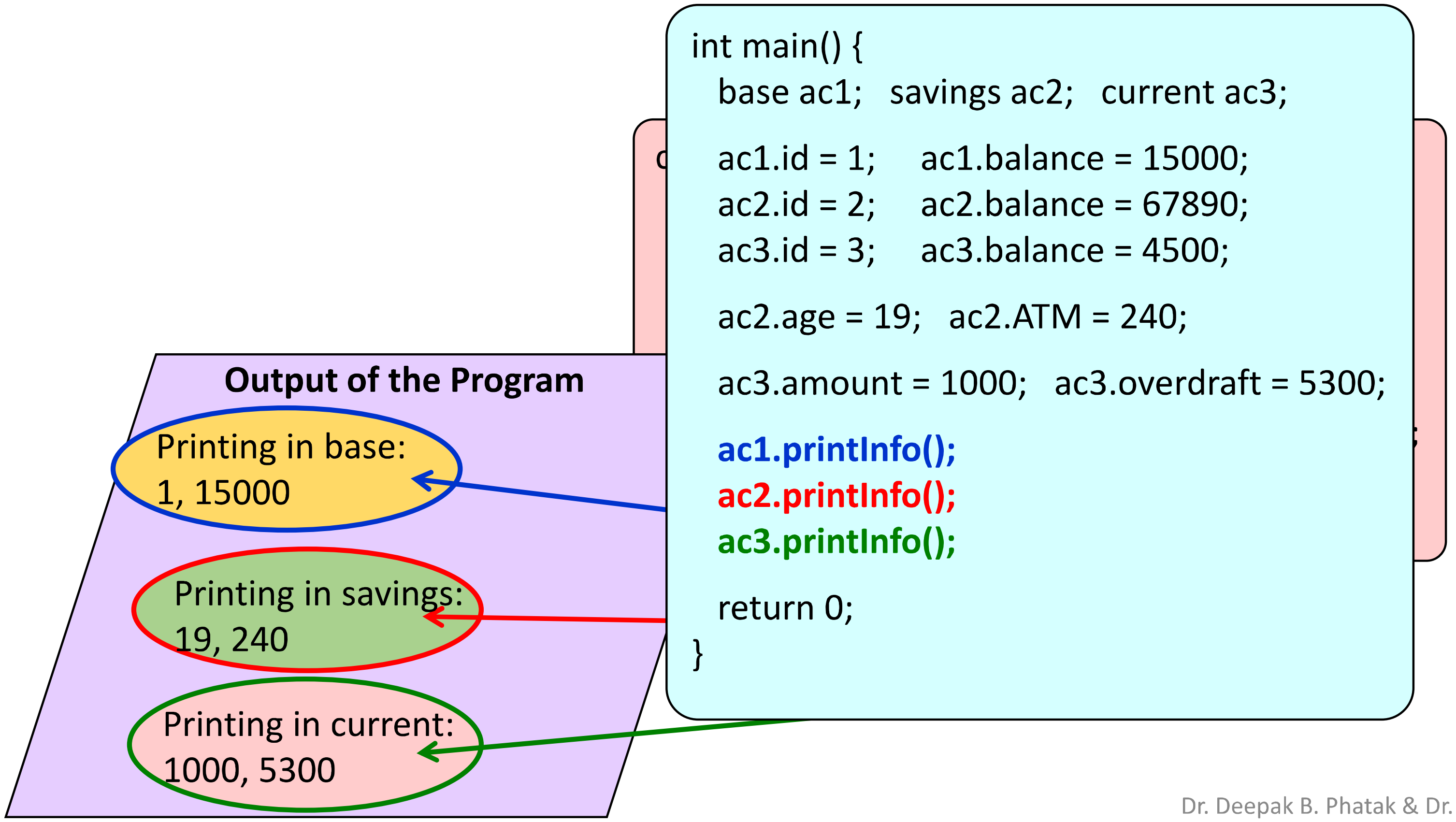
  McGraw Hill Education 2014

  Some examples used in this lecture are from the above book

Recall: Modified Bank Account hierarchy



Redefining member functions of base class



```
int main() {
  base ac1;   savings ac2;   current ac3;

  ac1.id = 1;     ac1.balance = 15000;
  ac2.id = 2;     ac2.balance = 67890;
  ac3.id = 3;     ac3.balance = 4500;

  ac2.age = 19;   ac2.ATM = 240;

  ac3.amount = 1000;   ac3.overdraft = 5300;

  ac1.printInfo();
  ac2.printInfo();
  ac3.printInfo();

  return 0;
}
```

Output of the Program

```
Printing in base:
1, 15000

Printing in savings:
19, 240

Printing in current:
1000, 5300
```

Access methods of base class using derived class

```
class base {
  public:
    int id;
    float balance;
    char name[];
    void printInfo() {
      cout << "Printing in base: \n" ;
      cout << id << ", " << balance << endl;
    }
};
```

```
class savings : public base {
  public:
    int age;
    long int ATM;
    void printInfo() {
      base :: printInfo();
      cout << "\nPrinting in savings: \n";
      cout << age << ", " << ATM << endl;
    }
};
```

Insert

```
class current : public base {
  public:
    int amount;
    int overdraft;
    void printInfo() {
      base :: printInfo();
      cout << "\nPrinting in current: \n";
      cout << amount << ", " << overdraft << endl;
    }
};
```

What, if we want to access **printInfo() of the base** class using **derived classes**

Output of the Program	Modified Output
Printing in base: 1, 15000	**Printing in base: 1, 15000**
Printing in savings: 19, 240	**Printing in base: 2, 67890 Printing in savings: 19, 240**
Printing in current: 1000, 5300	**Printing in base: 3, 4500 Printing in current: 1000, 5300**

```
int main() {
  base ac1;   savings ac2;   current ac3;

  ac1.id = 1;     ac1.balance = 15000;
  ac2.id = 2;     ac2.balance = 67890;
  ac3.id = 3;     ac3.balance = 4500;

  ac2.age = 19;   ac2.ATM = 240;

  ac3.amount = 1000;   ac3.overdraft = 5300;

  ac1.printInfo();
  ac2.printInfo();
  ac3.printInfo();

  return 0;
}
```

Constructors for Derived Classes

Case1: (a) With default constructor for base class.
(b) No explicit base constructor invocation in derived class

```
class base {
  public:
    int id;
    float balance;
    char name[];
    base(){ 2  7
3     cout << "Default constructor: base\n" ;
8     id = 0;
      balance = 0.0;
    };
    void printInfo() {
5     cout << "Printing in base: \n" ;
      cout << id << ", " << balance << "\n";
    }
};
```

```
int main() {
  base ac1;  1
  ac1.printInfo(); 4
  int age = 20;
  int ATM = 240;
  savings ac2(age, ATM); 6
  ac2.printInfo(); 11
  return 0;
}
```

```
class savings : public base {
  public:
    int age;
    long int ATM;
6a  savings(int x, int y): age(x), ATM(y){ 9
10    cout << "Derived constructor";
    }
12  void printInfo() {
      cout << "\nPrinting in savings: \n" ;
      cout << age << ", " << ATM << endl ;
    }
};
```

Output

```
Default constructor: base
Printing in base:
0, 0

Default constructor: base
Derived constructor
Printing in savings:
20, 240
```

Constructors for Derived Classes

Case2: (a) Without default constructor for base class (parameterised constructor)
(b) No explicit base constructor invocation in derived class

IIT Bombay

```
class base {
  public:
    int id;
    float balance;
    char name[];
    base(int a){  // constructor with argument
      cout << “Constructor: base\n";
      id = a;
      balance = 0.0;
    }
    void printInfo() {
      cout << "Printing in base: \n";
      cout << id << ", " << balance << “\n”;
    }
};
```

```
int main() {
  int ATM = 240;
  int age = 20;
  savings ac2(age, ATM);
  return 0;
}
```

Compile error

Will this program compile ?

```
class savings : public base {
  public:
    int age;
    long int ATM;
    savings(int x, int y): age(x), ATM(y){
      cout << "Derived co nstructor ";
    }
    void printInfo() {
      cout << "\nPrinti     n savings: \n";
      cout << age << "      < ATM << endl;
    }
};
```

expects base constructor to be invoked with an argument

Constructors for Derived Classes

Case3: (a) No default constructor for base class.
(b) Derived constructor specifying invocation of base constructor

```
class base {
  public:
    int id;
    float balance;
    char name[];
    base(int a) {        2  9
3     cout << "Default constructor: base\n" ;
10    id = a;
      balance = 0.0;
    };
    void printInfo() {
5     cout << "Printing in base: \n" ;
      cout << id << ", " << balance << “\n”;
    }
};
```

```
int main() {
  base ac1(1);          1
  ac1.printInfo();      4
  int id = 10, age = 20;
  int ATM = 240;
  savings ac2(id, age, ATM);   6
  ac2.printInfo();      12
  return 0;
}
```

```
class savings : public base {
  public:
    int age;
    long int ATM;
7   savings(int x, int y, int z):
                  8  base(x),
                  11 age(y), ATM(z) {
      cout << "Derived constructor";
    }
    void printInfo() {
13    cout << "\nPrinting in savings: \n" ;
      cout << age << ", " << ATM << endl ;
    }
};
```

Output

```
Default constructor: base
Printing in base:
1, 0

Default constructor: base
Derived constructor
Printing in savings:
20, 240
```

Constructors for Derived Classes

IIT Bombay

Case4: (a) No default constructor for base class.
(b) Initialize members of derived class: Using body

```
class base {
  public:
    int id;
    float balance;
    char name[];
    base(int a) {   2  9
3    cout << "Default constructor: base\n" ;
     id = a;
10   balance = 0.0;
    };
    void printInfo() {
5    cout << "Printing in base: \n" ;
     cout << id << ", " << balance << “\n”;
    }
};
```

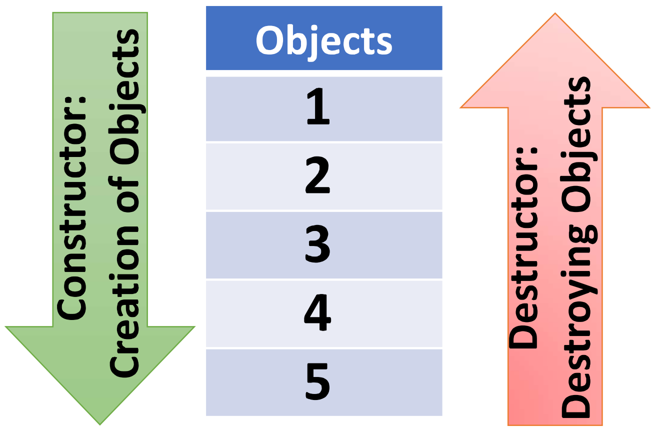
```
int main() {
  base ac1(1);  1
  ac1.printInfo();  4
  int id = 10, age = 20;
  int ATM = 240;
  savings ac2(id, age, ATM); 6
  ac2.printInfo();  12
  return 0;
}
```

```
class savings : public base {
  public:
    int age;
    long int ATM;                                   8
 7  savings(int x, int y, int z): base(x) {
      age = y;
      ATM = z;  11
      cout << "Derived constructor";
    }
13  void printInfo() {
      cout << "\nPrinting in savings: \n" ;
      cout << age << ", " << ATM << endl ;
    }
};
```

Output

```
Default constructor: base
Printing in base:
1, 0
Default constructor: base
Derived constructor
Printing in savings:
20, 240
```

Destructors



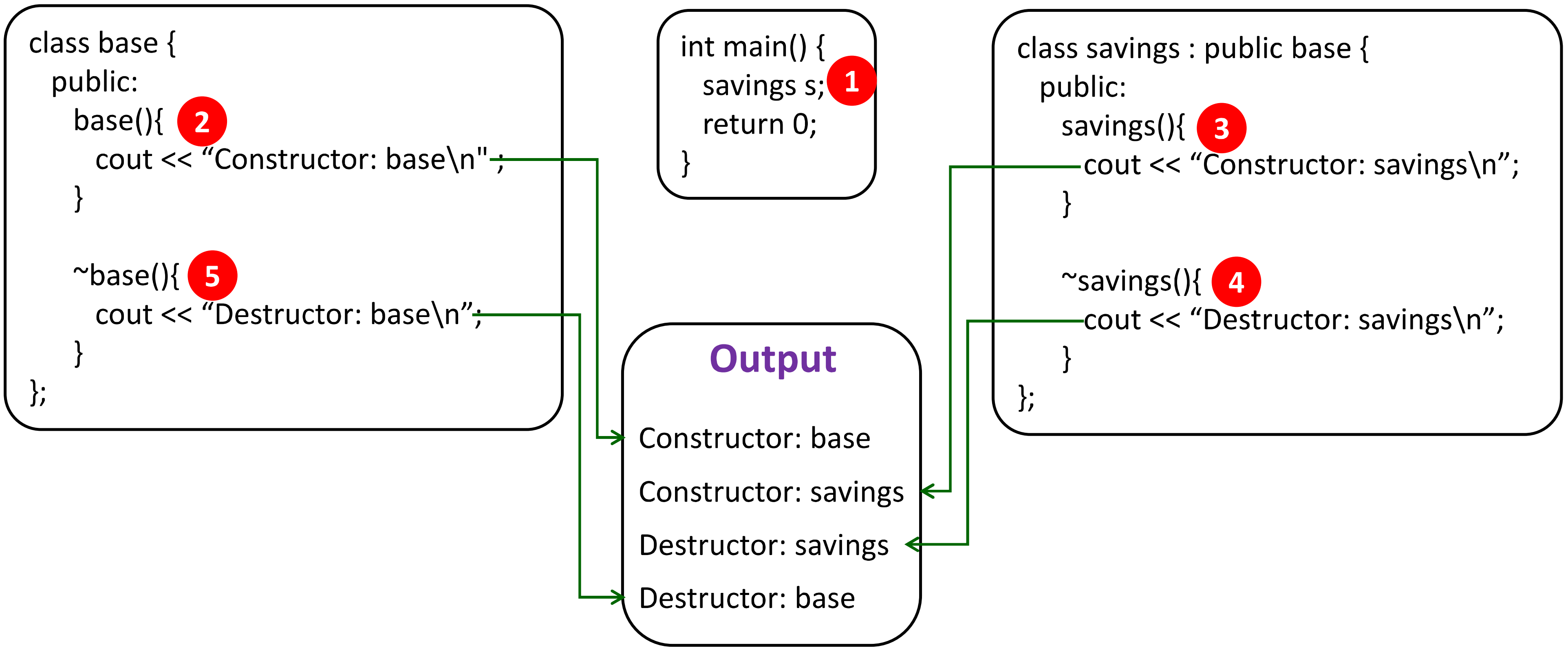
```
class A {
  public:

    A () {  // Constructor
       …
    }

    ~A() {
       …
    }

}
```

Destructors



```
class base {
  public:
    base(){
      cout << “Constructor: base\n" ;
    }

    ~base(){
      cout << “Destructor: base\n”;
    }
};
```

```
int main() {
  savings s;
  return 0;
}
```

```
class savings : public base {
  public:
    savings(){
      cout << “Constructor: savings\n”;
    }

    ~savings(){
      cout << “Destructor: savings\n”;
    }
};
```

Output

```
Constructor: base
Constructor: savings
Destructor: savings
Destructor: base
```

Inheritance of assignment operators

IIT Bombay

```
class base {
  public:
    int id;
    base(int x):id(x){ }   base constructor
    base & operator=(base &a){
       id = a.id;            assignment operator
       cout << "base class operator\n" ;
       return *this;
    }
};
```

```
class savings : public base {
  public:
    int age;                 savings constructor
    savings(int x, int y):base(x),age(y) { }
};
```

```
int main() {
  base b1(10);
  savings s1(11,20), s2(12,30);
  b1 = s1;  ✓
  s2 = b1;  ✗
  return 0;
}
```

b1.operator=(s1);

s2.operator=(b1): assignment operator is not inherited

Summary

- Redefining member functions of the base class
- Access methods of base class using derived classes
- Constructors for derived classes
- Destructors
- Inheritance of assignment operators