Method hiding

Method hiding (1)

```
struct Base {
  (virtual) void f(bool i)
            {cout << "f(bool)" << endl;}
  void f(int x, int y) {cout<<”f(two ints)”<<endl;}
};

struct Derived: public Base {
  void f(int b) {cout << "f(int)" << endl;}
};

int main(){
  Derived d;
  d.f(3);     // prints ‘f(int)’
  d.f(true); // prints ‘f(int)’
  d.f(4,5);   // prints ‘f(two ints)’
}
```

Method hiding (2)

```
struct Base {
  (virtual) void f(bool i)
            {cout << "f(bool)" << endl;}
};

struct Derived: public Base {
  void f(int b) {cout << "f(int)" << endl;}
};

int main(){
  Derived d;
  d.Base::f(true); // prints ‘f(bool)’
  d.Base::f(3);    // prints ‘f(bool)’
}
```

Method hiding (3)

```
struct Base {
  (virtual) void f(bool i)
            {cout << "bool" << endl;}
};

struct Derived: public Base  {
  using Base::f;
   void f(int b) {cout << "int" << endl;}
};

int main(){
   Derived d;
   d.f(3);     // prints ‘int’
   d.f(true); // prints ‘bool’
}
```

Multiple inheritance and virtual base class

Multiple inheritance

++C

- A class can inherit from multiple classes:

```
struct inputFile{
  void read();
};
struct outputFile{
  void write();
};
struct ioFile : public inputFile, public outputFile{
  …
};
// in main
ioFile f;
f.read();
f.write();
```

Multiple inheritance order

++C

- Construction and destruction order are according to the inheritance list:

```
struct inputFile{
  inputFile(){cout<<“inputFile ctor ”;}
};
struct outputFile{
  outputFile(){cout<<“outputFile ctor ”;}
};
struct ioFile: public inputFile, public outputFile{
  ioFile(){cout<<“ioFile ctor ”;}
};
// in main
ioFile f;//prints: inputFile ctor outputFile ctor ioFile ctor
```

Multiple inheritance

++C

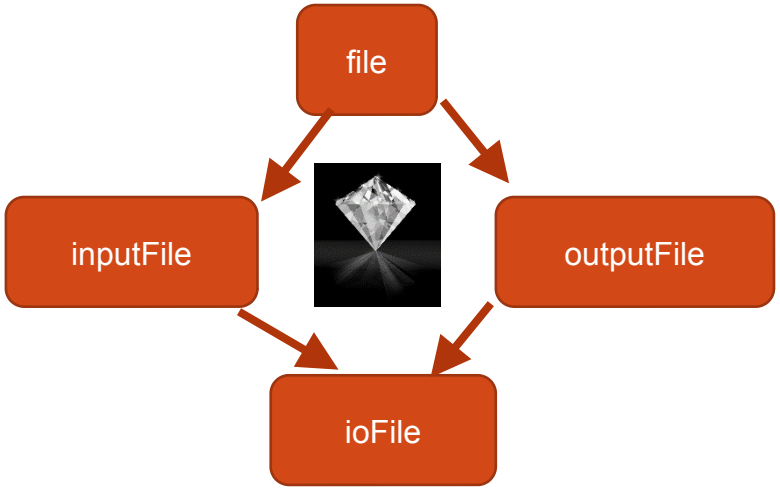
- Name ambiguities will generate compile error.
- In the following example ioFile has **two instances** of *open()*

```
struct inputFile{
  void open();
};
struct outputFile{
  void open();
};
struct ioFile: public inputFile, public outputFile{
  …
};
// in main
ioFile f;
f.open(); //error!
f.inputFile::open(); //Ok!
```

Diamond Multiple Inheritance

++C

```
struct file{
  char* name;
  void open();
}
struct inputFile : public file{
  void read();
};
struct outputFile : public file{
  void write();
};
struct ioFile: public inputFile, public outputFile{};
// in main
ioFile f;
f.open(); //error!
f.inputFile::open(); //Ok!
```



** ioFile still has **two instances** of *open()*

Diamond Multiple Inheritance

++C

```
struct file{
  char* name;
  void open();
}
struct inputFile : public file{
  void read();
};
struct outputFile : public file{
  void write();
};
struct ioFile: public inputFile, public outputFile{};
// in main
ioFile f;
f.name= “fileA.txt”; // error!
f.inputFile::name=“fileA.txt”; // Ok!
f.outputFile::name=“fileB.txt”; // Ok! Does not change
                                inputFile::name
```

Virtual Inheritance

++C

```
struct file{
  char* name;
  void open();
}
struct inputFile : virtual public file {
  void read();
};
struct outputFile : virtual public file {
  void write();
};
struct ioFile: public inputFile, public outputFile {};
// in main
ioFile f;
f.open(); // Ok!
          // ioFile has one instance of open() and name
```

Virtual inheritance: The base construction problem

```
struct file{
  file(char* name){…}
  char* _name;
struct inputFile: virtual public file{
  inputFile(char* name):file(name){}
};
struct outputFile: virtual public file{
  outputFile(char* name):file(name){}
};
struct ioFile: public inputFile, public outputFile{
  ioFile(char* name):inputFile(name),outputFile(name){}
};
```

Problem: File ctor will be initialized twice!

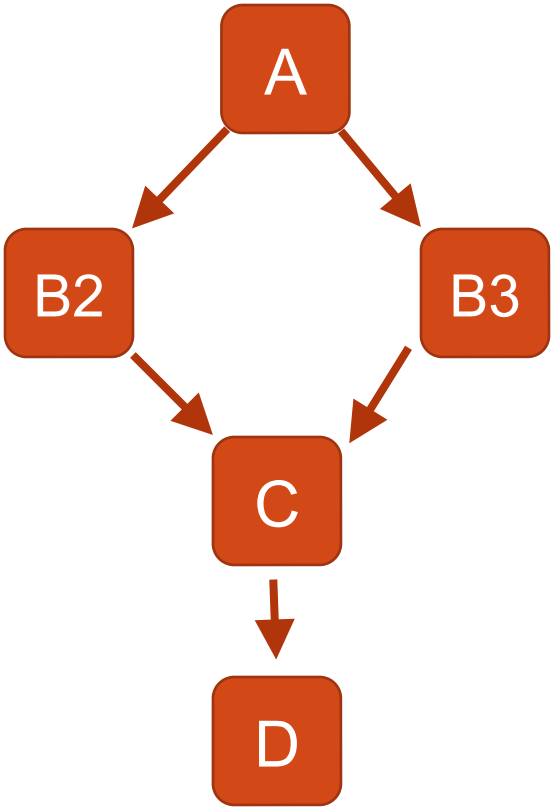
Virtual inheritance – the solution:

```
struct file{
  file(char* name){…}
  char* _name;
struct inputFile: virtual public file{
  inputFile(char* name):file(name){}
};
struct outputFile: virtual public file{
  outputFile(char* name):file(name){}
};
struct ioFile: public inputFile, public outputFile{
  ioFile(char* name):file(name),
 inputFile(name),outputFile(name){}
};
```

Solution: the base class is initialized by the most derived class

Virtual Base Class - D has to initialize A !

constructors for virtual base classes anywhere in your class's inheritance hierarchy are called by the "most derived" class's constructor



Interim Summary

- A known problem, easy to misuse.
- Usually restrict yourself to “interface like” multiple inheritance:
  - <=1 "real" base and
  - >=0 "interface" like (only pure virtual functions (no data members and no implementation)

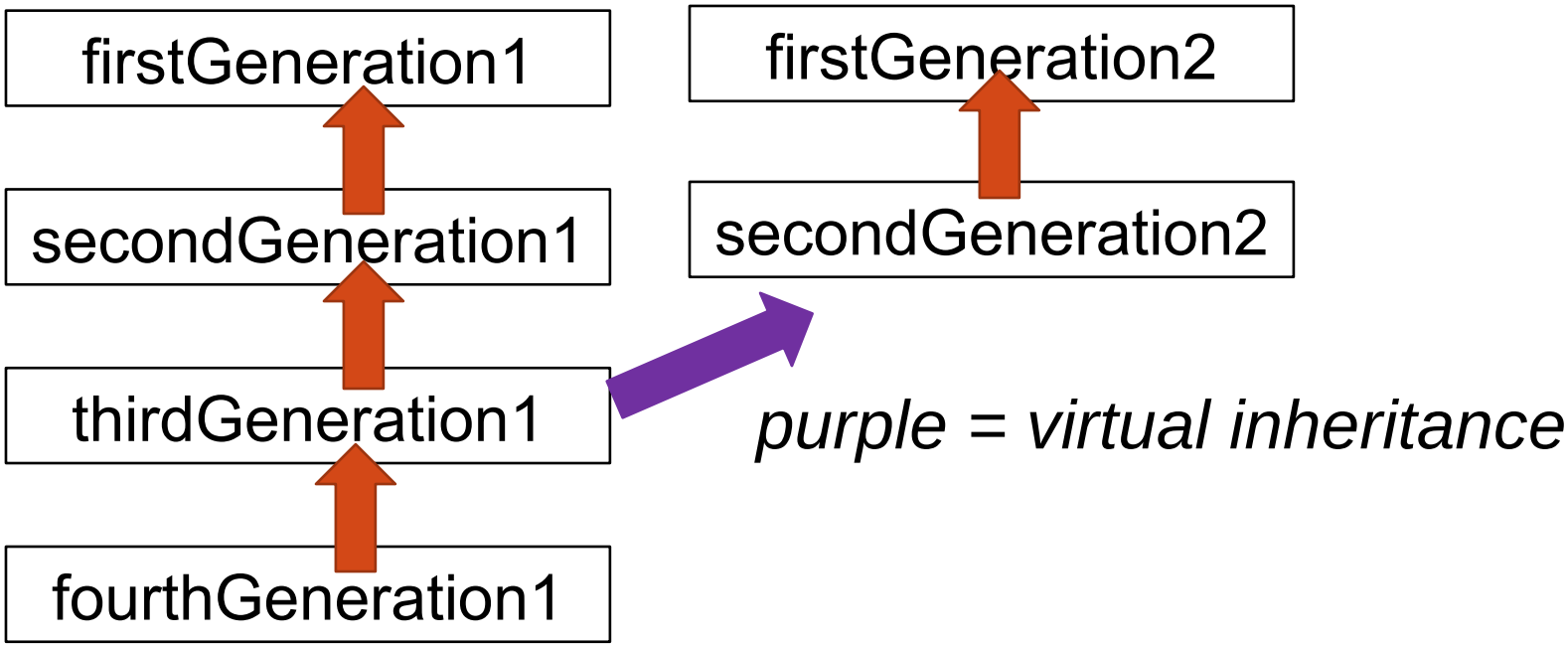
C-tors execution order

- virtual base classes anywhere in the hierarchy.
  - They are executed in the order they appear in a depth-first left-to-right traversal of the graph of base classes, where *left to right* refer to the order of appearance of base class names
- After all virtual base class constructors are finished:
  from base class to derived class.
  - The order is determined by the order that the **base classes appear in the declaration of the class**, *not* in the order that the initializer appears in the derived class's initialization list (compilers often give warnings).

Consider the following case:

```
struct firstGeneration1{
    firstGeneration1(){cout<<"first gen1\n";}
};
struct firstGeneration2{
    firstGeneration2(){cout<<"first gen2\n";}
};
struct secondGeneration1:public firstGeneration1{
    secondGeneration1(){cout<<"snd gen 1\n";}
};
struct secondGeneration2:public firstGeneration2{
    secondGeneration2(){cout<<"snd gen 2\n";}
};
struct thirdGeneration1: public secondGeneration1, virtual public secondGeneration2{
    thirdGeneration1(){cout<<"thirdGeneration1\n";}
};
struct fourthGeneration1: public thirdGeneration1{
    fourthGeneration1(){cout<<"fourthGeneration1\n";}
};
int main()
{
    fourthGeneration1 f;
```

The inheritance graph:



- fourthG1 calls secondG2 calls firstG2
- fourthG1 calls thirdG1 calls secondG1 calls firstG1
- Output is:

  firstG2, secondG2, firstG1, secondG1, thirdG1, fourthG1