Inheritance (+smart pointers, constructors with inheritance)

Smart Pointers:

Why use a smart pointer? Simply put:

- It takes care of allocating and freeing memory for you

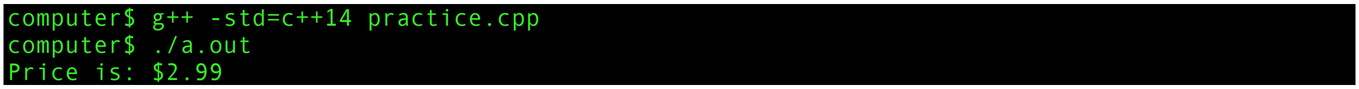
Two main types I will use: *unique* and *shared*

- Unique: only one pointer can point at an object (single owner of object)
- Shared: multiple pointers can point at an object

Example 1:

```
#include <iostream>
#include <memory> //included for smart pointers
#include <vector>

using namespace std;

class Phone{

};

int main(int argc, char **argv)
{
  string answer;
  unique_ptr<Phone> p_ptr=make_unique<Phone>();
  //unique_ptr<Phone> p2=p_ptr; //you can't do this

  shared_ptr<Phone> p_ptr1=make_shared<Phone>();
  shared_ptr<Phone> p3=p_ptr1; //you can do this
}
```

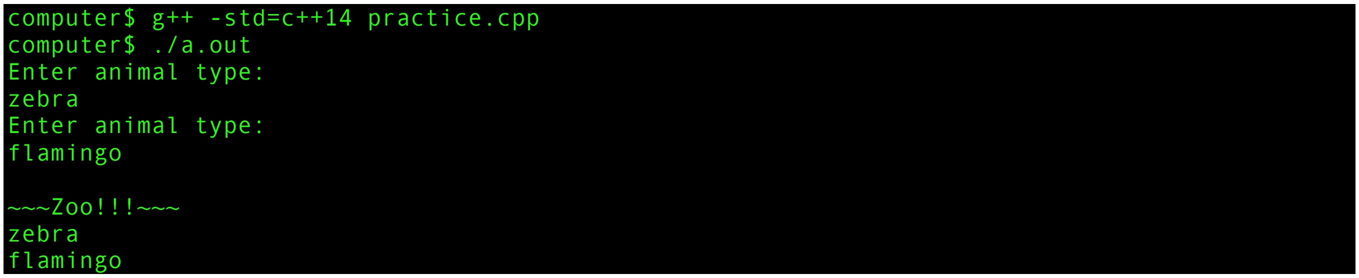
Example 2:



```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Price is: $2.99
```

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

class Cake{

  float price;

public:
  void set_price(float price)
  {
    this->price=price;
  }

  void print_price()
  {
    cout<<"Price is: $"<<price<<endl;
  }


};


class Bakery{

public:
  //create dessert-this function returns a unique_ptr
  unique_ptr<Cake> make_cake(float price)
  {
    unique_ptr<Cake> cake_ptr=make_unique<Cake>();
    cake_ptr->set_price(price);
    return cake_ptr;
  }

};

//if you return an object using new, you still need to remember to delete it when you're done
//smart pointers help you with this

int main (int argc, char **argv ()
{
    Bakery b1;
    unique_ptr<Cake> ckptr=b1.make_cake(2.99);
    ckptr->print_price();

}
```

Example 3:



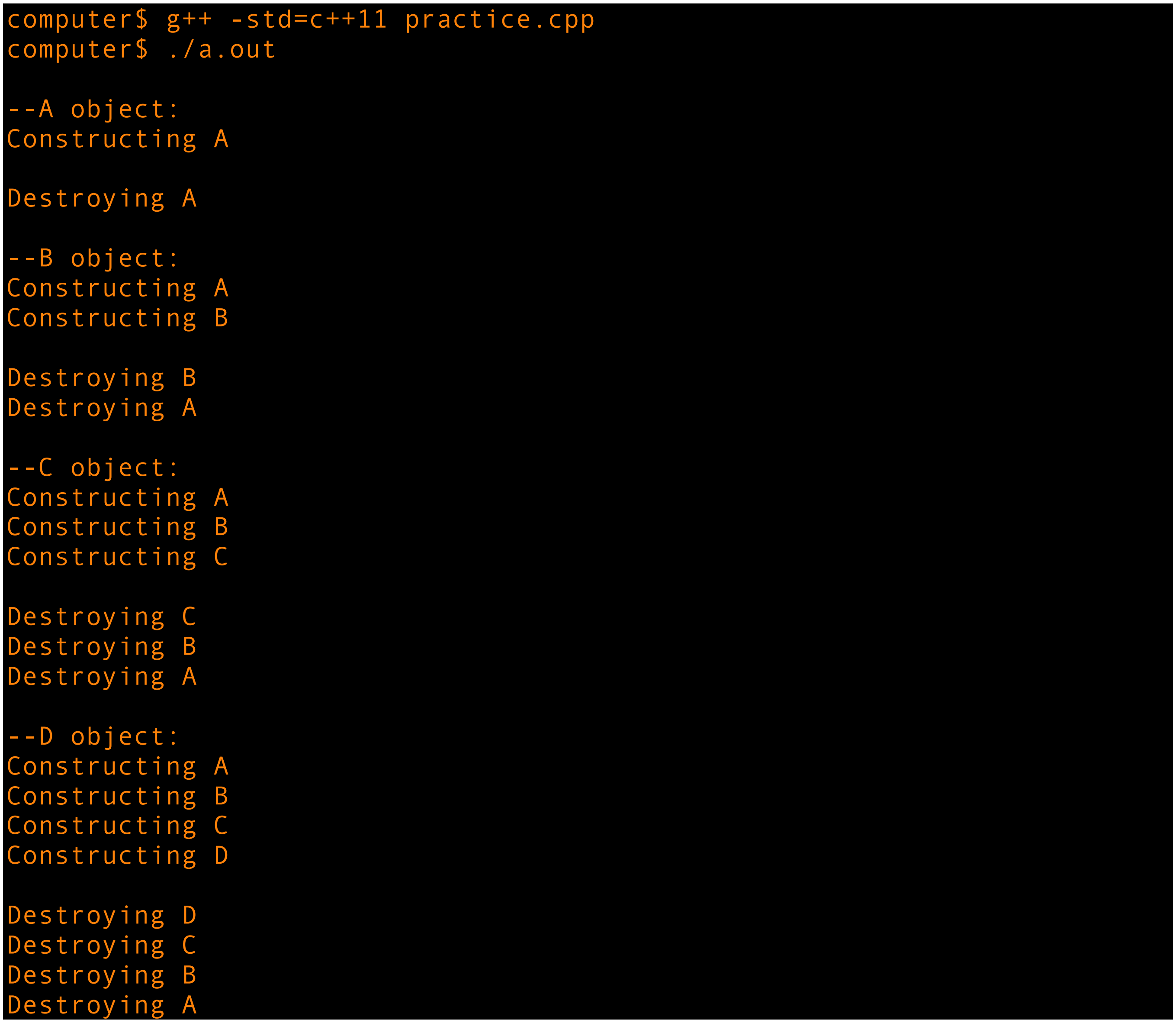
```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Enter animal type:
zebra
Enter animal type:
flamingo

~~~Zoo!!!~~~
zebra
flamingo
```

```
#include <iostream>
#include <memory>
#include <vector>

using namespace std;

class Animal{

public:
  string type;
  Animal(string name)
  {
    type=name;
  }
};


class Zoo{

  public:

  vector<shared_ptr<Animal>> all_animals;

   void new_animal()
   {
     string answer;
     cout<< "Enter animal type: "<<endl;
     cin >> answer;

     shared_ptr<Animal> p = std::make_shared<Animal>(answer);
     all_animals.push_back(p);
   }

   void show_zoo()
   {
      cout<<"\n~~~Zoo!!!~~~"<<endl;

     for(int i=0;i<all_animals.size();i++)
     {
       cout<< (all_animals[i]->type)<<endl;
     }
   }

};



int main(int argc, char **argv)
{
  Zoo z1;
```

```
 z1.new_animal();
 z1.new_animal();

 z1.show_zoo();
}
```

Using constructors with Inheritance:

First, notice then when creating an object from a derived class, the constructor for the base class is also called (also the order the destructors are called when the object goes out of scope):

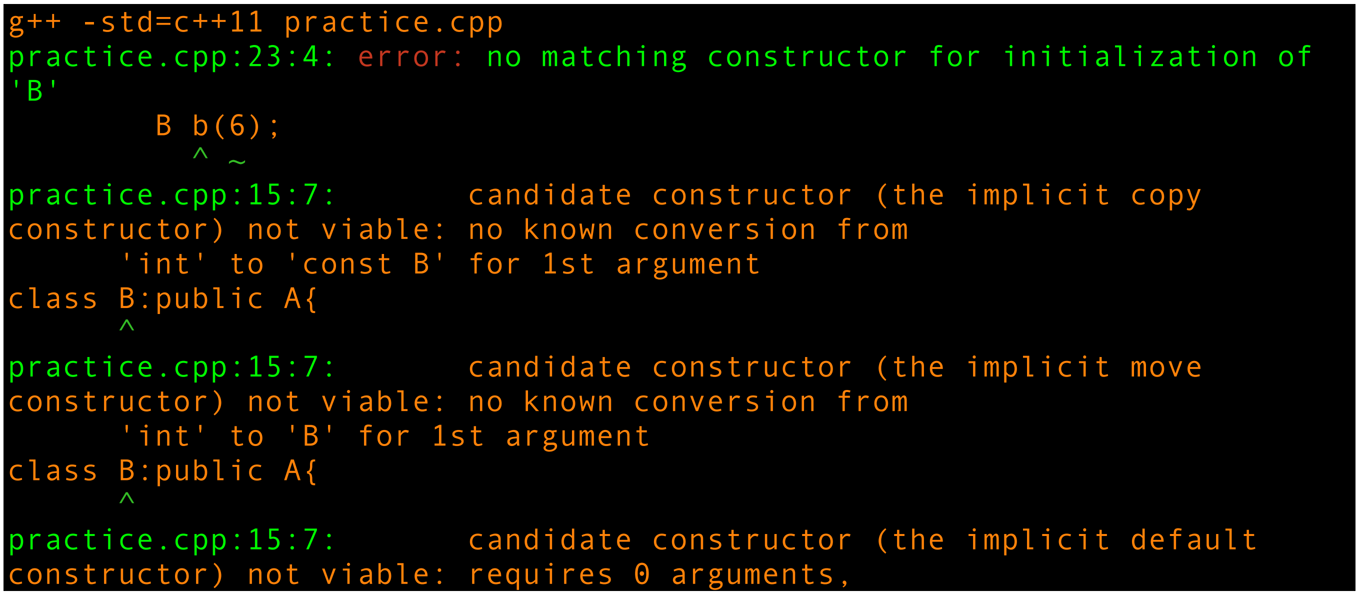


```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out

--A object:
Constructing A

Destroying A

--B object:
Constructing A
Constructing B

Destroying B
Destroying A

--C object:
Constructing A
Constructing B
Constructing C

Destroying C
Destroying B
Destroying A

--D object:
Constructing A
Constructing B
Constructing C
Constructing D

Destroying D
Destroying C
Destroying B
Destroying A
```

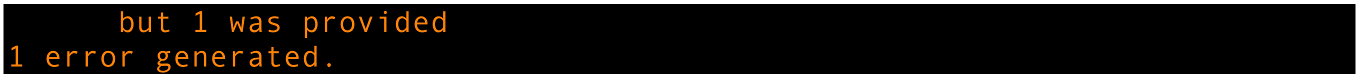
```
#include <iostream>

class A
{
public:
  A()
  {
     std::cout << "Constructing A\n";
   }

 ~A()
 {
 std::cout << "Destroying A\n";
 }
};

class B: public A
{
public:
 B()
 {
 std::cout << "Constructing B\n";
 }

 ~B()
 {
 std::cout << "Destroying B\n";
 }
};

class C: public B
{
public:
 C()
 {
 std::cout << "Constructing C\n";
 }

 ~C()
 {
 std::cout << "Destroying C\n";
 }
};

class D: public C
{
public:
 D()
 {
 std::cout << "Constructing D\n";
 }

 ~D()
 {
 std::cout << "Destroying D\n";
 }
};
```

```
int main(int argc, char **argv)
{
 if(true)
 {
 std::cout << "\n--A object:"<<std::endl;
   A cA;
 std::cout << "\n";
 }

 if(true)
 {
 std::cout << "\n--B object:"<<std::endl;
 B cB;
 std::cout << "\n";
 }

 if(true)
 {
 std::cout << "\n--C object:"<<std::endl;
 C cC;
 std::cout << "\n";
 }

 if(true)
 {
 std::cout << "\n--D object:"<<std::endl;
 D cD;
 std::cout << "\n";
 }

}
```

Now that we understand the preceding information, look at the following error:



```
g++ -std=c++11 practice.cpp
practice.cpp:23:4: error: no matching constructor for initialization of
'B'
        B b(6);
          ^ ~
practice.cpp:15:7:       candidate constructor (the implicit copy
constructor) not viable: no known conversion from
      'int' to 'const B' for 1st argument
class B:public A{
      ^
practice.cpp:15:7:       candidate constructor (the implicit move
constructor) not viable: no known conversion from
      'int' to 'B' for 1st argument
class B:public A{
      ^
practice.cpp:15:7:       candidate constructor (the implicit default
constructor) not viable: requires 0 arguments,
```



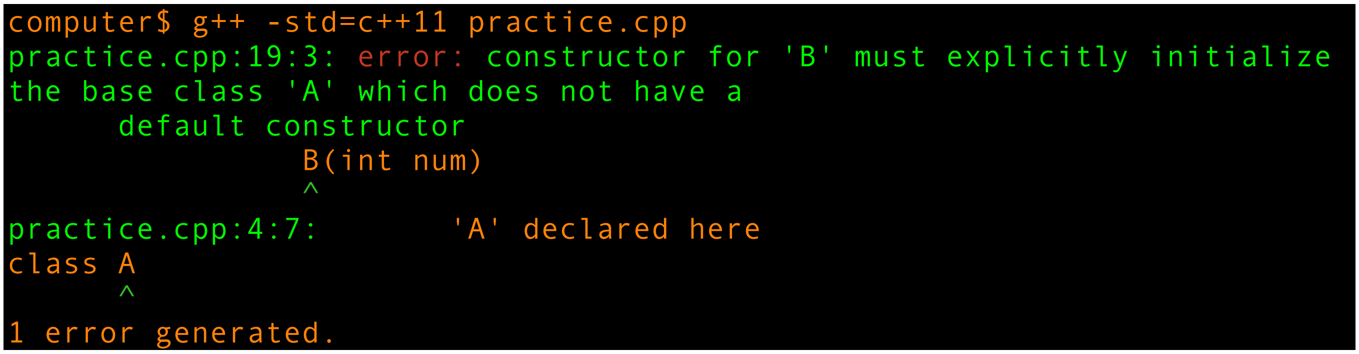
```
      but 1 was provided
1 error generated.
```

Why does this happen? Can the derived class B not "access" the constructor in base class A?

```
#include <iostream>

class A
{
 int a;
public:
    A(int num)
    {
        std::cout<<"Making A..."<<std::endl;
        a=num;
    }

};

class B:public A{

};


int main(int argc, char **argv)
{

 B b(6); //we get an error by including this line.  It appears that we do not have access to the constructor in A

}
```

What if we do this (put a constructor in B)?

-Nope, still an error. Remember you are calling the A constructor because we making an A object behind the scenes (but not giving it something):



```
computer$ g++ -std=c++11 practice.cpp
practice.cpp:19:3: error: constructor for 'B' must explicitly initialize
the base class 'A' which does not have a
      default constructor
                 B(int num)
                 ^
practice.cpp:4:7:       'A' declared here
class A
      ^
1 error generated.
```

```
#include <iostream>

class A
{
 int a;
public:
    A(int num){
 std::cout<<"Making A..."<<std::endl;
 a=num;
 }

};

class B:public A{
 int num1;

public:
 B(int num)
 {
 num1=num;
 }

};


int main(int argc, char **argv)
{

 B b(6);

}
```

So what is going on? We are trying to create a B object, but the A class constructor still needs to be give a value (remember from our example above, there is an A object behind the scenes). We can do the following:

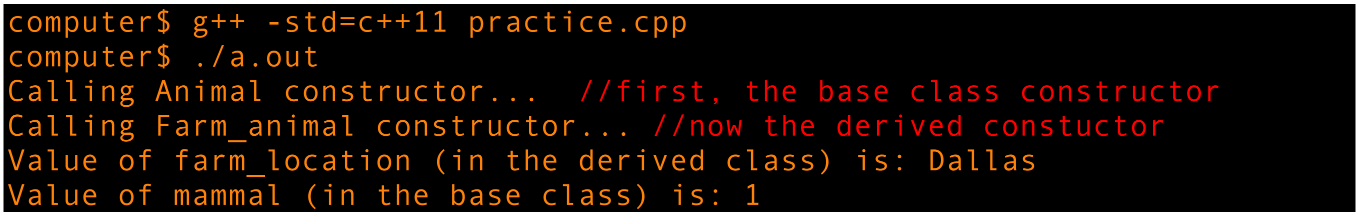
```
#include <iostream>

class A
{
 int a;
  public:
    A(int num){
    std::cout<<"Making A..."<<std::endl;
     a=num;
 }

};

class B:public A{
 int num1;

public:
 B(int n, int n2):A(n2) //the second parameter of the constructor is given to the A constructor
 {
 num1=n;
 }

};

int main(int argc, char **argv)
{

 B b(6,7);

}
```

Additionally, we could do the following:

```
#include <iostream>

class A
{
 int a;
public:
    A(int num){
 std::cout<<"Making A..."<<std::endl;
 a=num;
 }

 A() //We are adding an empty constructor here so we can now call this (not the one above) when making our A object behind the scenes
 {

 }

};

class B:public A{
 int num1;

public:
 B(int num)
 {
 num1=num;
 }

};


int main(int argc, char **argv)
{

 B b(6); //this if fine because it is first calling the B constructor (needing an int) then the empty A constructor we added

}
```

Example 1:

Create a constructor in the derived class



```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out
Calling Animal constructor...  //first, the base class constructor
Calling Farm_animal constructor... //now the derived constuctor
Value of farm_location (in the derived class) is: Dallas
Value of mammal (in the base class) is: 1
```

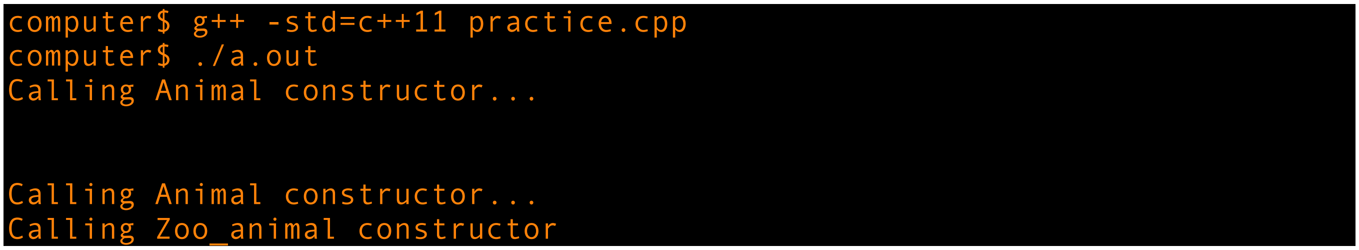
```
#include <iostream>

class Animal{
protected:
  bool mammal;

  Animal(bool mam)
  {
  std::cout<<"Calling Animal constructor..."<<std::endl;
  mammal=mam;

  }
};

class Farm_animal:public Animal{

  std::string farm_location;

public:
  Farm_animal(bool b, std::string city):Animal(b) //We have a Farm_animal (derived class) constructor that
takes two parameters.  One of the parameters (Boolean) is meant to be "fed" to the Animal (base class)
constructor
  {
     std::cout<<"Calling Farm_animal constructor..."<<std::endl;
     farm_location=city;
     std::cout<<"Value of farm_location (in the derived class) is: "<<farm_location<<std::endl;
     std::cout<<"Value of mammal (in the base class) is: "<<mammal<<std::endl;

  }
};


int main(int argc, char **argv) {
  Farm_animal f(true, "Dallas");
}
```

Note that if we had an empty constructor in Animal, we wouldn't need to worry about "feeding" the Animal constructor:

```
#include <iostream>

class Animal{
protected:
  bool mammal;

  Animal(bool mam)
  {
  std::cout<<"Calling Animal constructor..."<<std::endl;
  mammal=mam;

  }

  Animal() //because we have this, we don't need the Animal(b) below
  {

  }

};

class Farm_animal:public Animal{

  std::string farm_location;

public:
  Farm_animal(bool b, std::string city) //by not including the Animal (b) here, we are calling the empty constructor (we could still include it though if we wanted to call the Animal constructor with the boolean)
  {
     std::cout<<"Calling Farm_animal constructor..."<<std::endl;
     farm_location=city;
     std::cout<<"Value of farm_location (in the derived class) is: "<<farm_location<<std::endl;
     std::cout<<"Value of mammal (in the base class) is: "<<mammal<<std::endl;

  }
};

int main(int argc, char **argv) {
  Farm_animal f(true, "Dallas");
}
```

Example 2:

Accessing the base class constructor

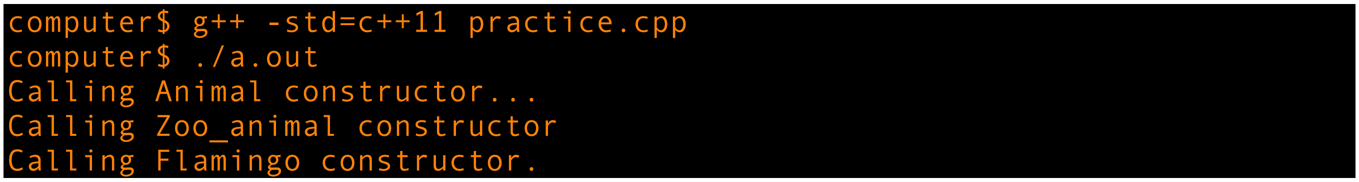


```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out
Calling Animal constructor...


Calling Animal constructor...
Calling Zoo_animal constructor
```

```
#include <iostream>

class Animal{
protected:
 bool mammal;

public:
  Animal(bool mam)
 {
 std::cout<<"Calling Animal constructor..."<<std::endl;
 mammal=mam;
 }
};

class Zoo_animal:public Animal{
 float price_ticket;

public:
 using Animal::Animal; //I now have access to the constructor in Animal

 Zoo_animal(bool b, float price):Animal(b)  //I also created a specific constructor for Zoo_animal
 {
 std::cout<<"Calling Zoo_animal constructor"<<std::endl;
 price_ticket=price;
 }

};

int main(int argc, char **argv) {

 Zoo_animal z(true); //using the Animal constructor
 std::cout<<"\n"<<std::endl;
 Zoo_animal z1(true, 2.99); //using the constructor used specifically for Zoo_animal

}
```

Example 3:

Constructors with multilevel inheritance



```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out
Calling Animal constructor...
Calling Zoo_animal constructor
Calling Flamingo constructor.
```

```
#include <iostream>
```

```
class Animal{
protected:
 bool mammal;

public:
  Animal(bool mam)
 {
 std::cout<<"Calling Animal constructor..."<<std::endl;
 mammal=mam;
 }
};

class Zoo_animal:public Animal{
 float price_ticket;

public:
 using Animal::Animal;

 Zoo_animal(bool b, float price):Animal(b)
 {
   std::cout<<"Calling Zoo_animal constructor"<<std::endl;
     price_ticket=price;
 }

};

class Flamingo:public Zoo_animal{
 int degree_pink;

public:
 Flamingo(int pink, bool bo, float p):Zoo_animal(bo,p) //only giving info to the class right above (not Animal)
 {
 std::cout<<"Calling Flamingo constructor."<<std::endl;
 degree_pink=pink;
 }
};

int main(int argc, char **argv) {

 Flamingo f1(2,false,2.99);
}
```

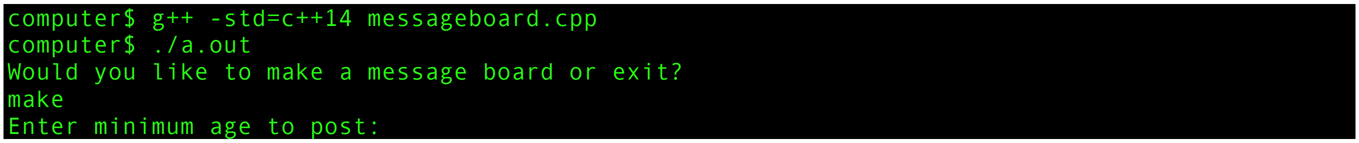
Example 4:

Constructors with multiple inheritance:

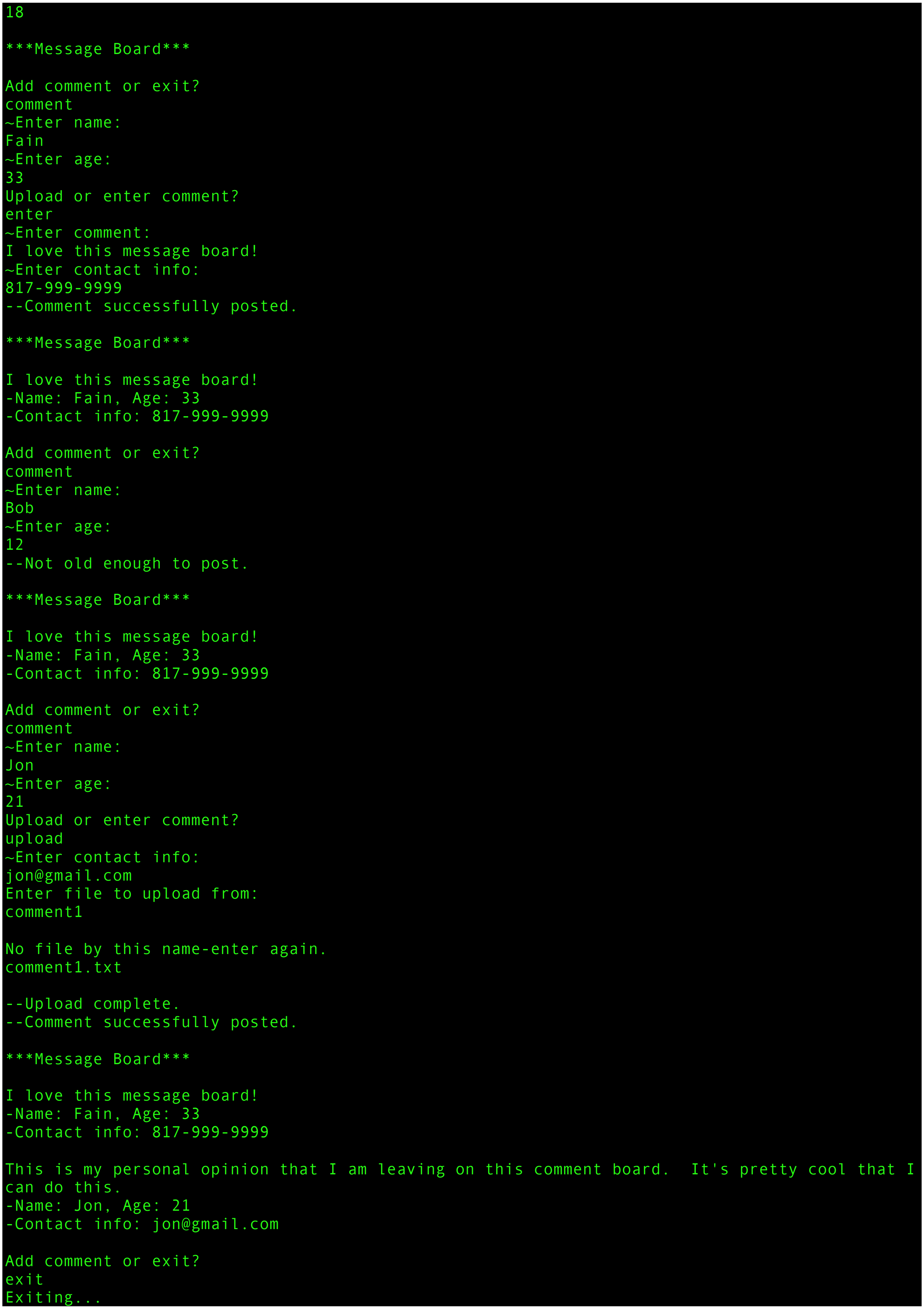
```
#include <iostream>

class Cup{
 std::string material;
public:
 Cup(std::string material)
 {
 this->material=material;
 }
};

class Tea{
 std::string flavor;

public:
 Tea(std::string flavor)
 {
 this->flavor=flavor;
 }


};


class Tea_cup:public Cup,public Tea{
 bool has_saucer;

public:
 Tea_cup(bool saucer, std::string flav, std::string material):Tea(flav),Cup(material) //notice I am giving info to
both base classes
 {
 has_saucer=saucer;
 }

};

int main(int argc, char **argv) {

 Tea_cup(true, "green_tea", "glass");

}
```

Program 1: Message Board

Create a message board. (I will use smart pointers for this)



```
computer$ g++ -std=c++14 messageboard.cpp
computer$ ./a.out
Would you like to make a message board or exit?
make
Enter minimum age to post:
```



```
18

***Message Board***

Add comment or exit?
comment
~Enter name:
Fain
~Enter age:
33
Upload or enter comment?
enter
~Enter comment:
I love this message board!
~Enter contact info:
817-999-9999
--Comment successfully posted.

***Message Board***

I love this message board!
-Name: Fain, Age: 33
-Contact info: 817-999-9999

Add comment or exit?
comment
~Enter name:
Bob
~Enter age:
12
--Not old enough to post.

***Message Board***

I love this message board!
-Name: Fain, Age: 33
-Contact info: 817-999-9999

Add comment or exit?
comment
~Enter name:
Jon
~Enter age:
21
Upload or enter comment?
upload
~Enter contact info:
jon@gmail.com
Enter file to upload from:
comment1

No file by this name-enter again.
comment1.txt

--Upload complete.
--Comment successfully posted.

***Message Board***

I love this message board!
-Name: Fain, Age: 33
-Contact info: 817-999-9999

This is my personal opinion that I am leaving on this comment board.  It's pretty cool that I
can do this.
-Name: Jon, Age: 21
-Contact info: jon@gmail.com

Add comment or exit?
exit
Exiting...
```

```
#include <iostream>
#include <fstream>
#include <memory>
#include <vector>

using namespace std;

class Person{

public:
  string name;
  string contact_info;
  int age;

  Person(string name, string contact_info, int age)
  {
    this->name=name;
    this->contact_info=contact_info;
    this->age=age;
  }
};

class Comment{

public:
      unique_ptr<Person> p_ptr;
      string comment;

  string type;
  Comment(string name, int age, string comment, string contact_info)
  {
    p_ptr=make_unique<Person>(name, contact_info, age);
    this->comment=comment;
  }
};

class Upload_comment:public Comment{

    using Comment::Comment;

public:
  string upload(string filename) //return comment
  {
    string fileinput;
    ifstream inFile;
    inFile.open(filename);

    while(!inFile.is_open()) //keep entering filename until correct one is given
```

```
{
     string answer;
     cout << "\nNo file by this name-enter again."<<endl;
     cin>>answer;
     inFile.open(answer);
  }

  getline(inFile, fileinput);
  cout<<"\n-Upload complete."<<endl;
  return fileinput;
 }


};


class Message_board{

vector<shared_ptr<Comment>> board;
int age;

public:

 Message_board(int age)
 {
  this->age=age;
 }

 int add_comment() //1 successful, -1 not successful
 {
  string name, comment, info, answer;
  int age;

  cout << "~Enter name:"<<endl;
  cin >> name;

  cout << "~Enter age:"<<endl;
  cin>> age;

  if (age < this->age)
  {
   return -1; //not successful, no comment
  }

  else
  {
   cout<<"Upload or enter comment?"<<endl;
   cin>>answer;

   if(answer=="enter")
   {
        cout << "~Enter comment:"<<endl;
        getchar();
        getline(cin,comment);

        cout << "~Enter contact info:"<<endl;
        cin>> info;

        shared_ptr<Comment> ptr=make_shared<Comment>(name, age, comment, info); //create a comment
        board.push_back(ptr);
        return 1;
      }

      else //assume upload
      {
        cout << "~Enter contact info:"<<endl;
        cin>> info;

        shared_ptr<Upload_comment> ptr=make_shared<Upload_comment>(name, age, "none yet", info);
//create a comment
        cout<<"Enter file to upload from:"<<endl;
        cin>>answer;
        ptr->comment=ptr->upload(answer);
        board.push_back(ptr);
        return 1;
      }
    }
  }

  void print_board()
  {
    cout<< "\n***Message Board***\n"<<endl;

    for(int i=0;i<board.size();i++)
    {
      cout<<board[i]->comment<<endl;
      cout<<"-Name: "<<board[i]->p_ptr->name<<", Age: "<<board[i]->p_ptr->age<<endl;
      cout<<"-Contact info: "<<board[i]->p_ptr->contact_info<<"\n"<<endl;
    }

  }
};

int main(int argc, char **argv)
{
  string answer;
  unique_ptr<Message_board> m_board;

  cout<<"Would you like to make a message board or exit?"<<endl;
  cin>>answer;

  if(answer=="make") //make board
  {
    cout<<"Enter minimum age to post:"<<endl;
    cin>>answer;

    m_board=make_unique<Message_board>(stoi(answer));

    while(answer!="exit")
    {
      m_board->print_board();

      cout<<"Add comment or exit?"<<endl;
      cin>>answer;

      if(answer=="comment")
      {
        int n=m_board->add_comment();

        if(n==-1)
        {
          cout<<"--Not old enough to post."<<endl;
        }

        else
        {
          cout<<"--Comment successfully posted."<<endl;
        }
      }
    }
  }

  cout<<"Exiting..."<<endl;

}
```