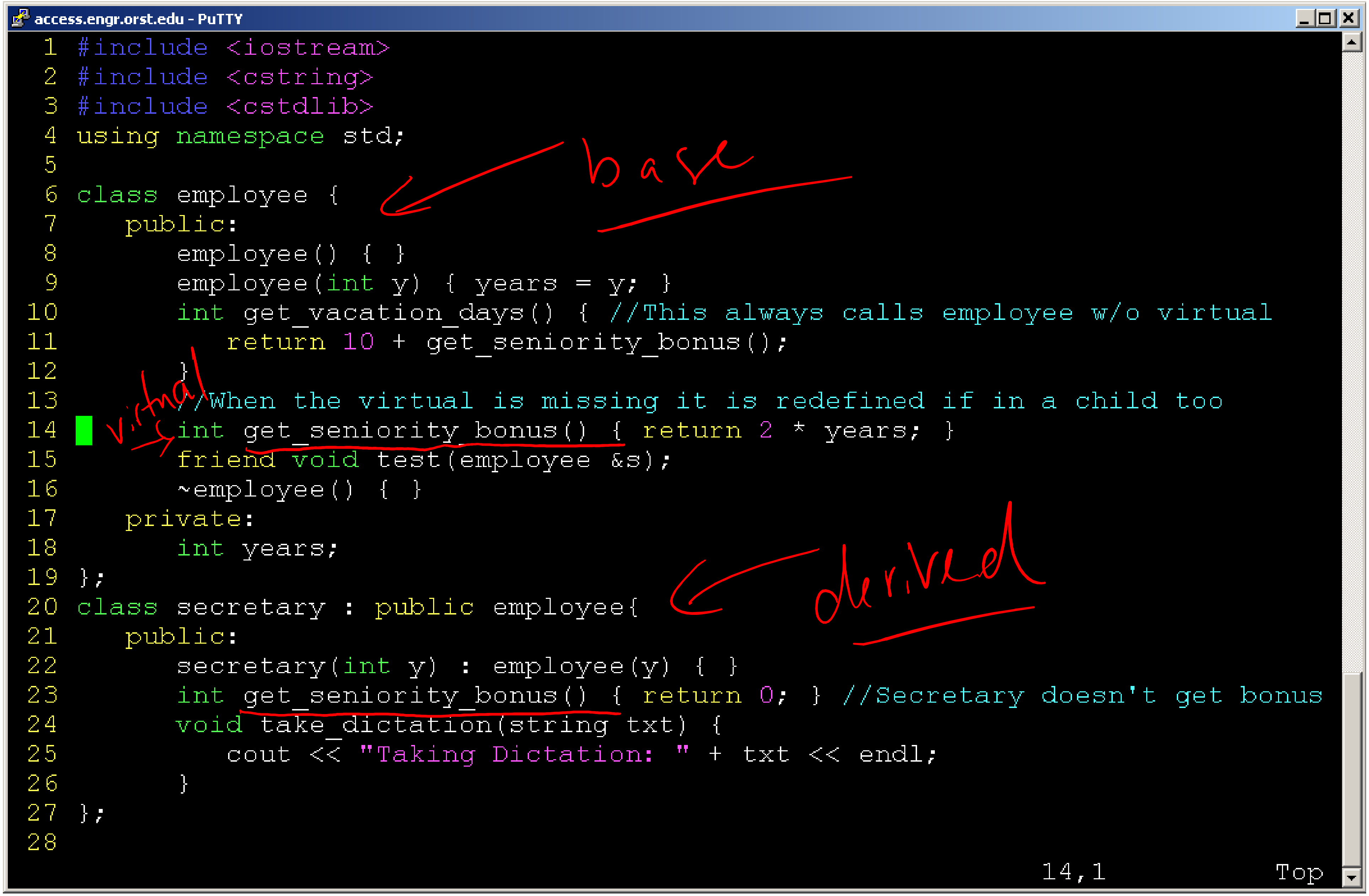
CS 162
Intro to CS II

Finish Polymorphism

Redefine vs. Polymorphism



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

class employee {
   public:
      employee() { }
      employee(int y) { years = y; }
      int get_vacation_days() { //This always calls employee w/o virtual
         return 10 + get_seniority_bonus();
      }
      //When the virtual is missing it is redefined if in a child too
      int get_seniority_bonus() { return 2 * years; }
      friend void test(employee &s);
      ~employee() { }
   private:
      int years;
};
class secretary : public employee{
   public:
      secretary(int y) : employee(y) { }
      int get_seniority_bonus() { return 0; } //Secretary doesn't get bonus
      void take_dictation(string txt) {
         cout << "Taking Dictation: " + txt << endl;
      }
};

                                                                  14,1          Top
```

What is polymorphism?

base derived

- Vehicle, Bike example…
- Revisit our code

Extending Types/Polymorphism

late binding
- runtime
- dynamic
- ptrs

- Can upcast, but not down

  Parent p; Child c;

  p = c; //what will the polymorph function call now?

  ~~c = p~~

- What if we made pointers?

  Parent *p; Child *c = new Child;

  p = c;

  Child c;

  p = &c

2 pts. Extra Credit

Make Destructors Virtual Tip

Child c;

- What does this do if destructor isn't virtual?

```
Child *c=new Child;  vs. Parent *p = new Child;
delete c;
delete p;
```

destructors called → Child then Parent

if no virtual, only parent destructor called.

- **Example:**

```
class parent {
  public:
    parent() {   //Have a constructor
      shared_ptr = new int;
    }
    virtual ~parent() {  //Have a destructor
      delete shared_ptr;
    }
private:
    int *shared_ptr;
};
```

if there is dynamic mem in Child

Oregon State University

Pure Virtual

- Definition
  - Don't need to define function in base/parent class
  - Why?
- Abstract class ← you can't define an object of this type
  - One or more pure virtual functions in class

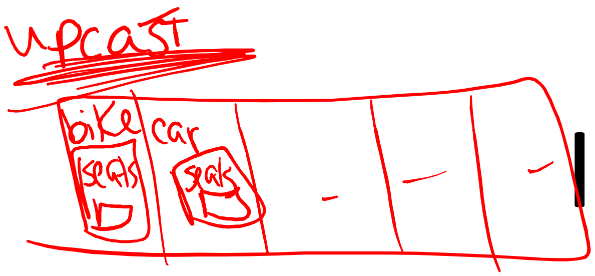
Pure Virtual

```
class figure {
public:
   figure();
   ~figure();
   virtual void draw() = 0;
   center() { … draw(); …}
};

class circle : public figure  {
public:
   circle();
   ~circle();
   void draw() { … }
};

class rectangle : public figure {
public:
   rectangle();
   ~ rectangle ();
   void draw() { … }
};
```

base

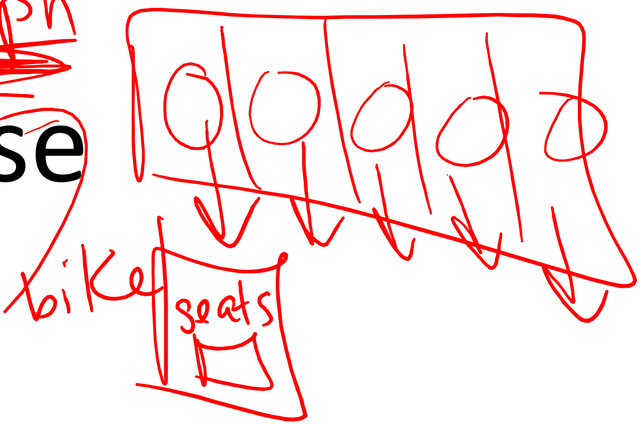
you do not define the functionality here.

When you have one or more pure virtual, then you have an abstract class



In Class Exercise

polymorph



- Get into groups 4-5.
- Discuss Lab#5: vehicle, bike, skateboard, car, motorcycle, and date classes.
  - What is the relationship?
  - What did you learn about polymorphism and upcasting?
  - Are any of these classes abstract classes?

"has a"

Date obj in vehicle class

Date obj in the main class

Date obj in the get_toll or they pass to get_toll to use

vehicle v[5] vs vehicle *v[5]

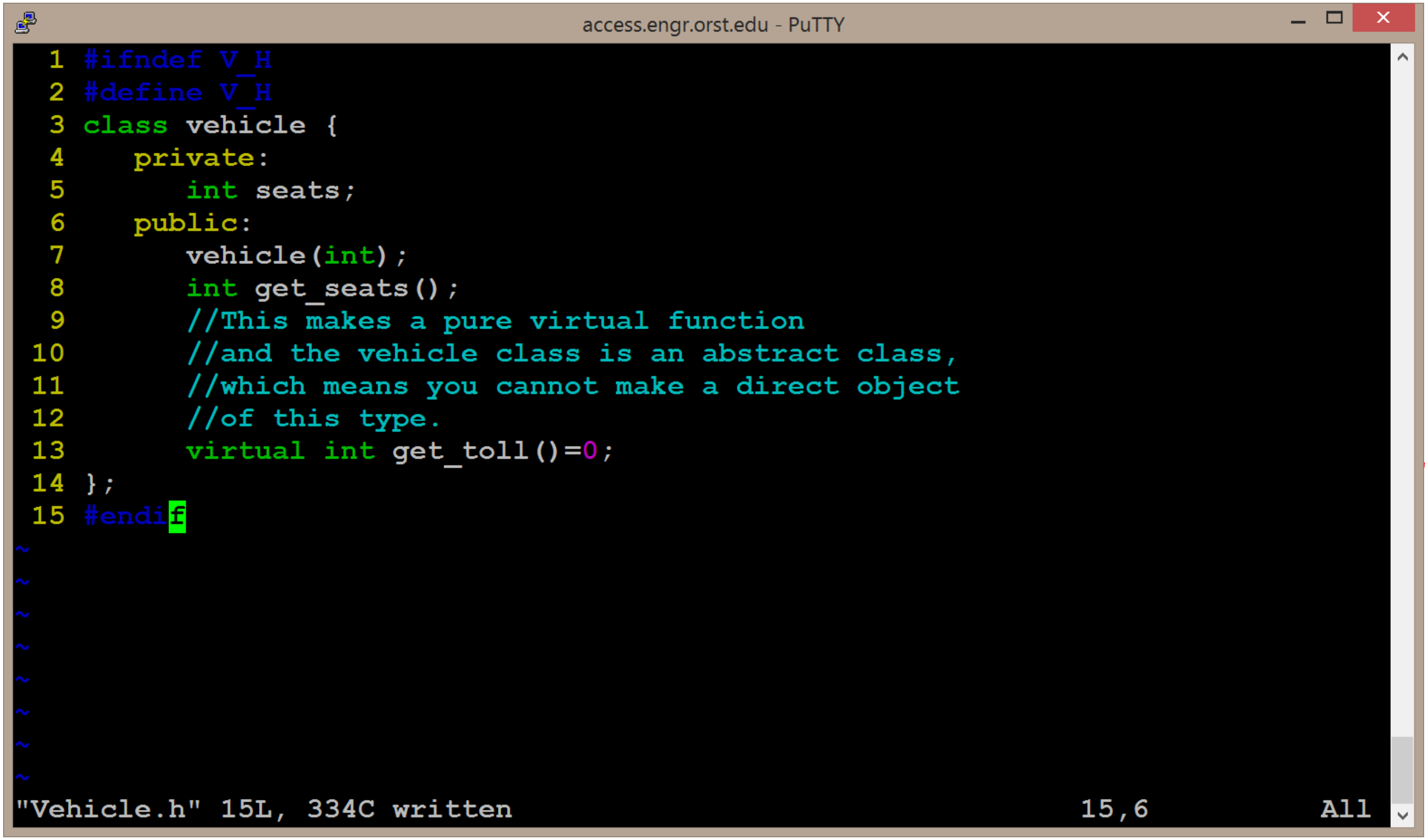
upcasting

v[0]=bike(1);

v[0]= new bike(1);

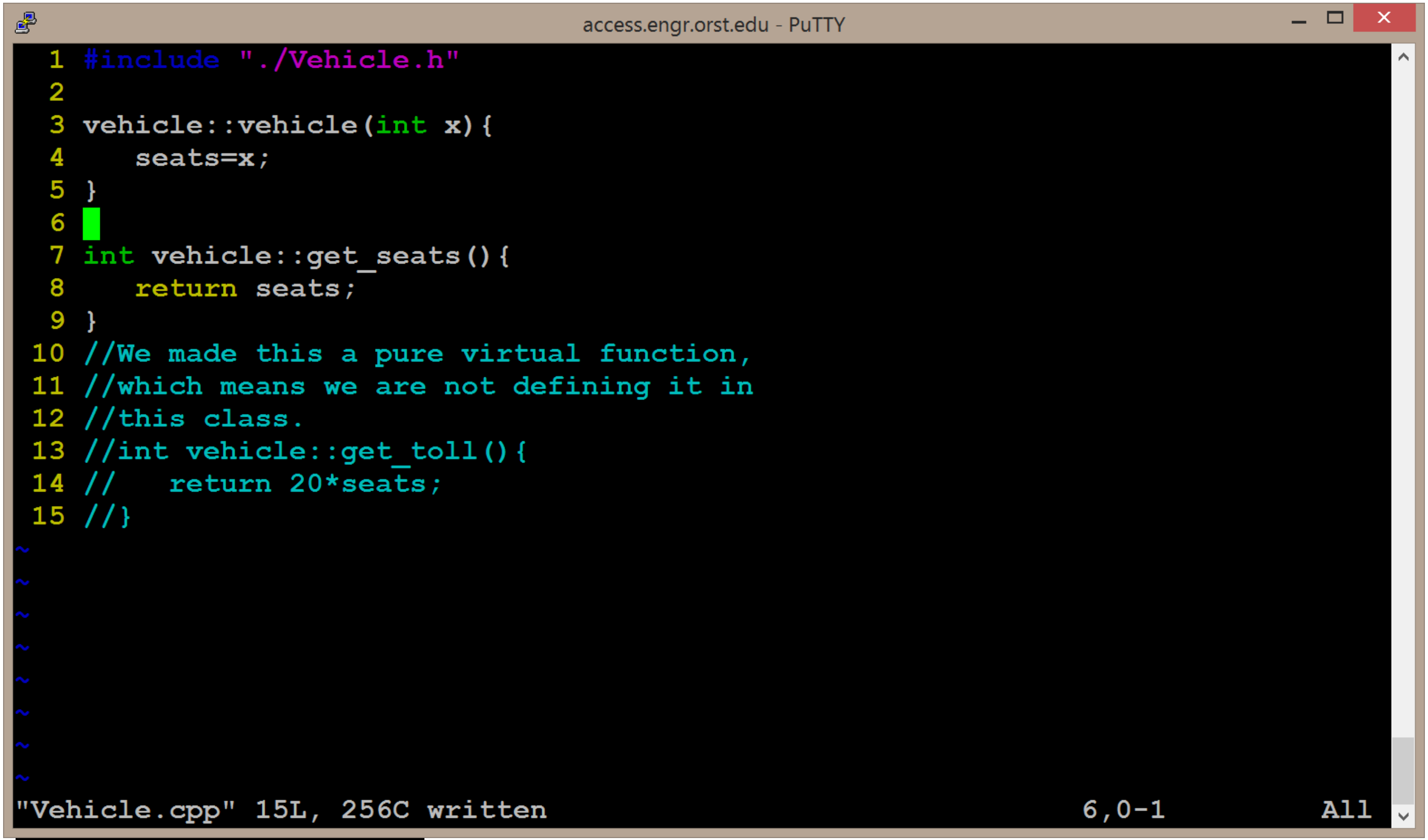
polymorph

Vehicle.h



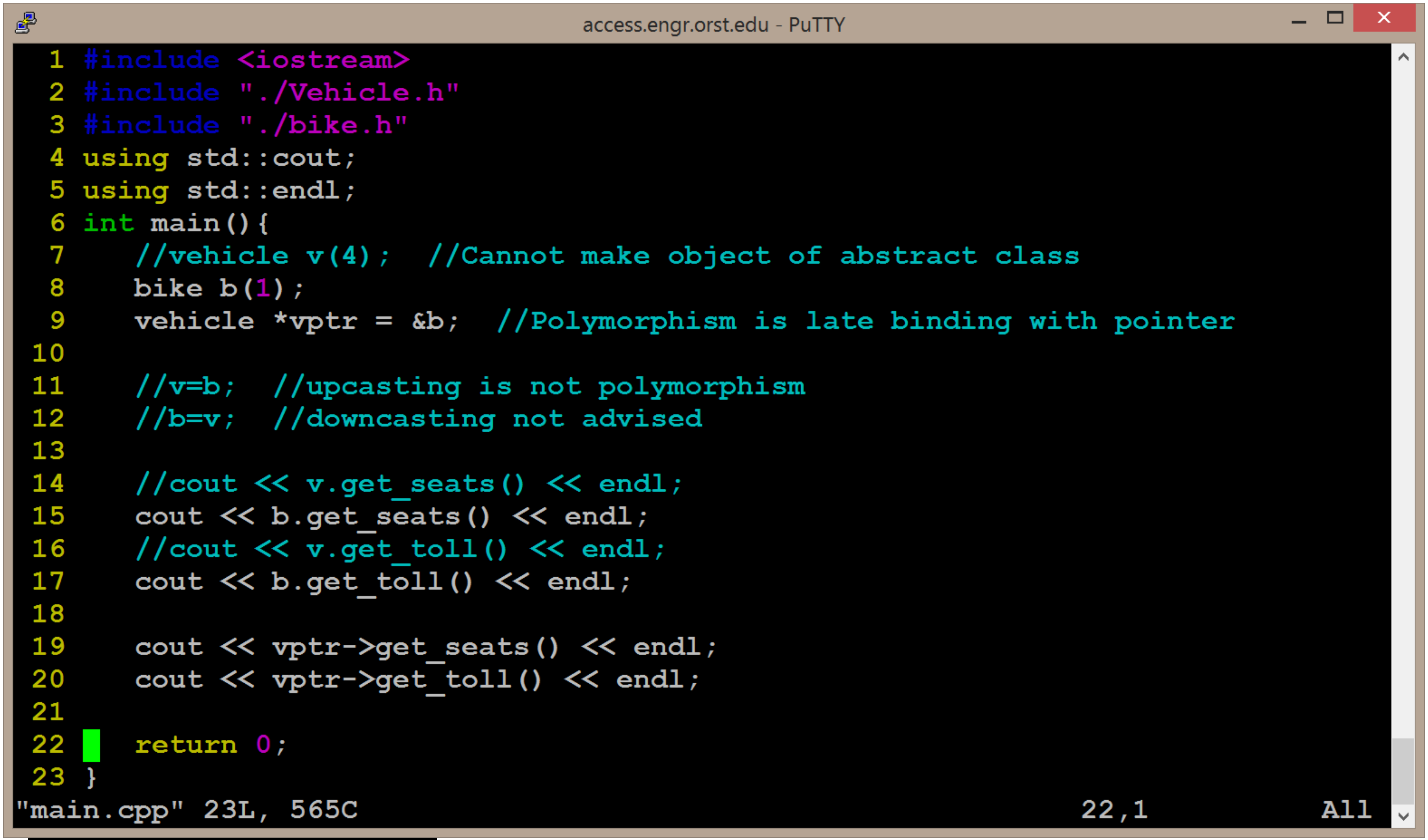
```
#ifndef V_H
#define V_H
class vehicle {
   private:
      int seats;
   public:
      vehicle(int);
      int get_seats();
      //This makes a pure virtual function
      //and the vehicle class is an abstract class,
      //which means you cannot make a direct object
      //of this type.
      virtual int get_toll()=0;
};
#endif
~
~
~
~
~
~
~
~
"Vehicle.h" 15L, 334C written                                   15,6          All
```

Vehicle.cpp



```
#include "./Vehicle.h"

vehicle::vehicle(int x){
   seats=x;
}

int vehicle::get_seats(){
   return seats;
}
//We made this a pure virtual function,
//which means we are not defining it in
//this class.
//int vehicle::get_toll(){
//   return 20*seats;
//}
~
~
~
~
~
~
~
~
"Vehicle.cpp" 15L, 256C written                              6,0-1         All
```

main.cpp



```
#include <iostream>
#include "./Vehicle.h"
#include "./bike.h"
using std::cout;
using std::endl;
int main(){
   //vehicle v(4);  //Cannot make object of abstract class
   bike b(1);
   vehicle *vptr = &b;  //Polymorphism is late binding with pointer

   //v=b;  //upcasting is not polymorphism
   //b=v;  //downcasting not advised

   //cout << v.get_seats() << endl;
   cout << b.get_seats() << endl;
   //cout << v.get_toll() << endl;
   cout << b.get_toll() << endl;

   cout << vptr->get_seats() << endl;
   cout << vptr->get_toll() << endl;

   return 0;
}
"main.cpp" 23L, 565C                                          22,1          All
```