Name: *(as it would appear on official course roster)*		
Umail address:	**@umail.ucsb.edu**	**section** 9am, 10am, 11am, 12pm
Optional: name you wish to be called if different from name above.		
Optional: name of "homework buddy" (leaving this blank signifies "I worked alone"		

1

h12

CS32 F19

h12: Polymorphism

ready?	assigned	due	points
true	Tue 11/05 02:00PM	Tue 11/12 02:00PM	50

You may collaborate on this homework with AT MOST one person, an optional "homework buddy".

MAY ONLY BE TURNED IN IN THE LECTURE LISTED ABOVE AS THE DUE DATE,
OR IF APPLICABLE, SUBMITTED ON GRADESCOPE. There is NO MAKEUP for missed assignments;
in place of that, we drop the lowest scores (if you have zeros, those are the lowest scores.)

Reading: Polymorphism, PS 15.3

1. (10 pts) Fill in the information in the header. The following are required to get the 10 "participation" points.
   - Filling in your name and umail address.

   Also: For paper submission PLEASE submit on ONE SHEET OF PAPER, double-sided if at all possible. If you must submit on two printed sheets write name on BOTH sheets and no staples, paperclips, or folded corners.

2. On p. 868 in PS, in Display 15.12, line 17, there is a use of the overloaded operator `<` on two objects, one of type `Sale` and another of type `DiscountSale`. The definition of that operator appears on lines 25 - 28 of Display 15.10 on p. 866. On line 27, there is an invocation of `first.bill()` and an invocation of `second.bill()`.

   a. (5 pts) For `first.bill()` in the case of the invocation in Display 15.12 line 17, where is the definition of the member function bill() that is invoked? Give the Display number, line number(s) and page number(s) of the textbook where it appears.

   b. (5 pts) For `second.bill()` in the case of the invocation in Display 15.12 line 17, where is the definition of the member function bill() that is invoked? Give the Display number, line number(s) and page number(s) of the textbook where it appears.

   c. (5 pts) The `bill()` member function is "special", meaning that the exact definition of the function used depends on what type of object it is invoked on - if it is an instance of `Sale` or `DiscountSale`, for example, which may not be known until run-time. What is the C++ keyword that is used in the source code on the definition of `bill()` that signals this so called *run-time dispatch* of the member function?

   d. (5 pts) Is `bill()` an example of a function with early binding or late binding?

2

h12

CS32 F19

3. (8 pts) Assume we have a base class (e.g. Person) and derived class (e.g. Student), and there is some function such as `toString()` that is defined in both the base class and the dervied class. For example, suppose that:

   - for Person, `toString` returns the person's name, e.g. `Chris Gaucho`
   - for Student, `toString` returns the person's name and their perm number in parentheses. e.g. `Chris Gaucho (1234567)`.

   We say that `toString()` is *overridding* in the derived class. However, in PS (15.3), Savitch makes a distinction between the two cases, one that is properly called *overriding* and another that should really be called *redefinition*. Most of the cases we've seen so far are really just *redefinition*. What is different, according to Savitch, in the case where this should be called *overriding*?

4. Given the following class definitions (you may assume all necessary libraries have already been included):

```
class A {
public:
    ~A() { cout << "A::~A()" << endl; }
    void f1() { cout << "A::f1()" << endl; }
    virtual void f2() { cout << "A::f2()" << endl; }
};
class B : public A {
public:
    virtual ~B() { cout << "B::~B()" << endl; }
    virtual void f1() { cout << "B::f1()" << endl; }
    void f2() { cout << "B::f2()" << endl; }
    virtual void f3() = 0;
};
class C : public B {
public:
    ~C() { cout << "C::~C()" << endl; }
    void f1() { cout << "C::f1()" << endl; }
    virtual void f3() { cout << "C::f3()" << endl; }
};
```

   a. (6 pts) What will the output be if we ran the following code (be sure to include destructors' output):

```
void f1() {
    C c1;
    A a1 = c1;
    a1.f1();
    a1.f2();
}

int main() { f1(); }
```

   b. (6 pts) What will the output be if we ran the following code (be sure to include destructors' output):

```
void f2() {
    B* b1 = new C();
    b1->f1();
    b1->f2();
    b1->f3();
    delete b1;
}

int main() { f2(); }
```