

# CSE399 Spring 2008 PRACTICE Exam SOLUTIONS

---

**Name:**

---

By signing below I swear or affirm that all answers I give on this exam represent my own individual knowledge and effort. I will neither receive nor give any improper help to other students during the administration of this exam.

---

**Signature:**

---

- There are 5 questions.
- This exam is 50 minutes long. Pace yourself accordingly.
- If you have any questions, please raise your hand.

Topic	Question	Points	Maxium
Pointers	1		20
Constructors/Operator Overloading	2		20
Templates	3		20
Object layout/inheritance	4		20
OO Designs/ STL	5		20
	Total		100

## Question 1: Pointers

**Part A (5 points):** Given the following declarations:

```
char c = 'A';
char * p = &c;
char ** p2 = &p;
void * v = &p2;
```

Examine each of the following expressions. If the expression is illegal, write ILLEGAL. If the expression is legal, write its type (i.e. `int`, `void *`, etc):

- `&p2` `char ***`
- `*p2` `char *`
- `&v` `void **`
- `p2 + 1` `char **`
- `v[0]` ILLEGAL (attempting to dereference a void \*)

**Part B (5 points):** Briefly explain what the function `realloc` does in C:

**Answer:**

The `realloc` function resizes a block of dynamically allocated memory. It takes in a pointer to the memory to resize, and the newly requested size. On success, `realloc` returns a pointer to the new memory, and the old pointer is invalid (as if it had been freed). On failure, `realloc` returns `NULL`. This function will typically try to resize the memory in place if it can, but if it cannot, it will copy the data to the newly allocated memory before freeing the old.

**Part C (10 points):** Write the function `swap`, which takes two pointers to integers and swaps the values pointed to by them. Show the proper way to call your `swap` function such that the `printf` statements print what the comments indicate:

**Answer:**

```
/* Fill in parameter list first */
void swap ( int * px, int * py ) {

    /* your code goes here */

    int temp = *px;

    *px = *py;

    *py = temp;

}

int main(void) {

    int a = 3;
    int b = 4;

    /* fill in arguments here */
    swap ( &a, &b );

    printf("a is now %d\n", a);    // should print a is now 4
    printf("b is now %d\n", b);    // should print b is now 3
    return 0;
}
```

## Question 2: Constructors/Operator Overloading

**Part A (3 points):** Copy constructors and the assignment operator (=) are similar. Copy constructors, however, are used only in specific circumstances: list the three situations in which copy constructors are used:

**Answer:**

The copy constructor is used when creating a new object from an old one of the same type. The three cases where this occurs are

1. Declaring a variable and initializing it from another expression of the same type in one statement.
2. Returning an object by value.
3. Passing an object as a parameter by value.

**Part B (6 points):** What is the appropriate signature for the copy constructor for a class called Foo?

**Answer:**

```
Foo::Foo (const Foo & rhs)
```

**Part C (6 points):** What is the appropriate signature for the assignment operator in the class called Foo (to allow assignment of one Foo to another)?

**Answer:**

```
Foo & Foo::operator= (const Foo & rhs)
```

**Part D (5 points):** There is an important difference between how dynamically dispatched functions behave during object construction in C++ and Java. Explain what this is and how it occurs.

**Answer:**

In Java, the vtables of an object are initialized to the proper vtable for the class being constructed immediately by the Object constructor. This means that if you call an overridden method in any of the constructors, it will dispatch to the method in the class being new'ed (i.e. the same method you would get on the resulting object after construction).

In C++, on the other hand, each constructor sets the object's vtable to the vtable of the type the object "is so far". This means that while in the parent object's super constructor, any functions will be called as if the object was only the parent type.

## Question 3: Templates

**Part A (10 points):** Write the function `findMin` which is templated over the type of item it processes. The `findMin` function should take a vector (i.e. the STL vector) of the given type. It should return the smallest item in the vector by value. You may assume the vector passed in is non-empty. Hint: you should use an iterator.

**Answer:**

```
template <class T>
T findMin(vector<T> vect) {
    typename vector<T>::iterator it = vect.begin();
    T x = *it;
    ++it;
    while (it != vect.end()) {
        T temp = *it;
        if (temp < x)
            x = temp;
        ++it;
    }
    return x;
}
```

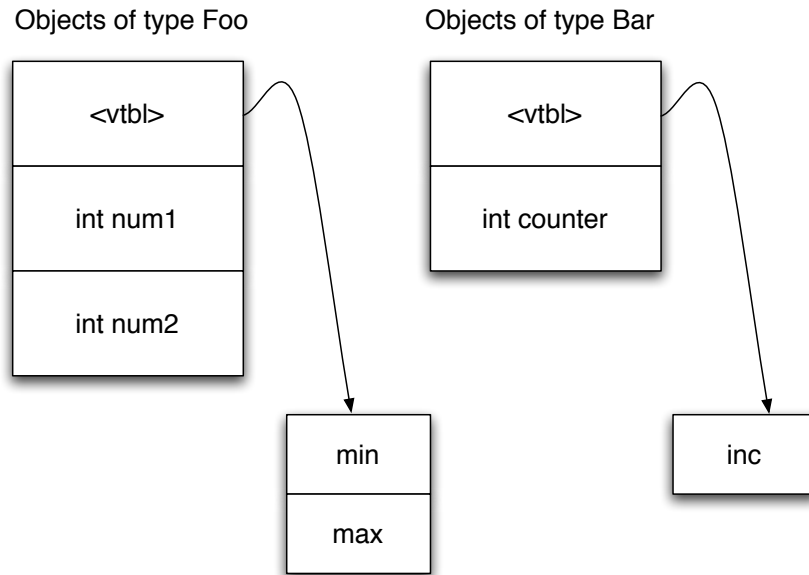
**Part B (10 points):** The function you wrote in part A uses the `<` operator, which is not defined on all types. Explain how the C++ compiler handles this. When/where would an error occur if you tried to use this function on a type for which `<` was not defined?

**Answer:**

The C++ compiler type checks a template only when it is instantiated (i.e. when it is used). This means that the compiler only checks if the less-than operator is defined on the types for which `findMin` is actually used. If you tried to instantiate `findMin` on a type that did not have a less than operator, the compiler would report the error for that instantiation (it would tell you where in the template body the less-than is, but first would tell you where the offending instantiation is).

## Question 4: Object Layout

For this question, use the following object layouts for objects of type Foo and Bar:

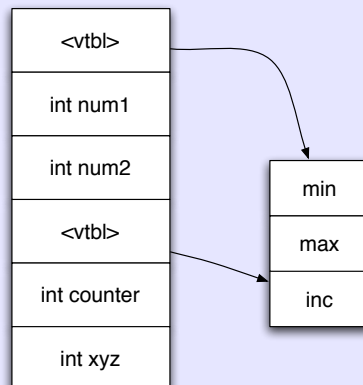


**Part A (10 points)** Class Baz is declared as follows:

```
class Baz: public Foo, public Bar {  
    int xyz;  
};
```

Draw the layout of an object of type Baz in the space below:

**Answer:**



**Part B (5 points)** Briefly explain what a vtable is and what is used for.

**Answer:**

A vtable is a table of function pointers that are used for dynamically dispatching functions. Each object that has at least one virtual function has as its first field a pointer to a vtable. When a virtual function is called, the pointer to the vtable in the object is followed, and the appropriate function pointer is read from the vtable. That function is then called.

**Part C (5 points)** Given the following declarations:

```
class A {...}
class B: public A {...}
class C: public A {...}
class D: public B, public C {...}
```

How many A sub-objects does a D have? Suppose that this is not the behavior you desired, indicate what changes need to be made to the above declarations to get the other possible behavior.

**Answer:**

A D object has two different A sub-objects. One inside its B sub-object, and one inside its C sub-object. It is possible to arrange the inheritance hierarchy for D such that it only has one A sub-object. Such an inheritance is called virtual inheritance and must be specified at B and C (class B: public virtual A) since B and C objects have to be laid out differently to support virtual inheritance.

## Question 5: OO Design and STL

**Part A (5 points):** For each statement, indicate true or false.

**false** - Methods should be noun-like.

**false** - Changing a good design to accommodate different features should be a massive undertaking: new changes are best redesigned from the ground up.

**false** - The “monolithic” design pattern is useful for minimizing the cognitive load on a code maintainer.

**false** - Computer scientists should seek to special case as many things as possible.

**false** - Design patterns are only useful in C++.

**Part B (8 points):** The C++ STL vector class has no method to remove a particular element from the vector. The STL provides a different means of removing a specific element. Briefly explain how this is accomplished.

**Answer:**

The vector class does not have a remove method, however, objects can be removed via iterators. The STL provides a remove algorithm which operates on any iterator so that it can be used with many different types of containers.

**Part C (7 points):** Name the design pattern shown below:

```
class A {
    virtual X* makeX() = 0;
    virtual Y* makeY() = 0;
};

class OneThing : public A {
    virtual X* makeX() {
        return new XOne();
    }
    virtual Y* makeY() {
        return new YOne();
    }
};
....
```

```
A* a = new OneThing();
.....
```

```
X * x = a->makeX();
...
```

**Answer:**

This design pattern is called Abstract Factory.