CS 211 Winter 2004
Sample Final Exam

Instructor: Brian Dennis, Assistant Professor
TAs: Tom Lechner, Rachel Goldsborough, Bin Lin

March 15, 2004

Your Name:

There are 6 problems on this exam. Each one is worth 20 points. They are not all of equal difficulty. Please read all of them before starting and attempt the least difficult first.

You have 120 minutes to complete the exam. **When time is up, please stop writing and turn in your quiz. If you have to be asked to stop you will receive no credit for the question you are working on.**

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Problem 5	
Problem 6	
Total	

Problem 1

10 Points. Explain why a static member function of a class shouldn't try to access a `this` variable.

10 points. Briefly give and explain one positive aspect and one negative aspect of operator overloading in C++.

Problem 2

```
-----------------------------------------------------------
// classes.h
class Base {
public:
  Base();
  virtual void howdy();
  void doody();
private:
  // ...
};

class Derived : public Base {
public:
  Derived();
  virtual void howdy();
  void doody();
private:
   //...
};
-----------------------------------------------------------
// base.cpp
#include <iostream>
#include "classes.h"
using namespace std;
// ...
void Base::howdy() {
  cout << "Let's get ready to CS rumble!" << endl;
};

void Base::doody() {
  cout << "I think I'm going to finish early." << endl;
};
-----------------------------------------------------------
// derived.cpp
#include <iostream>
#include "classes.h"
using namespace std;
// ...
void Derived::howdy() {
  cout << "I hope you were the groom!" << endl;
}

void Derived::doody() {
  cout << "Ted Nugent called. He wants his shirt back." << endl;
}
```

Assuming the code on the previous page, for each of the numbered lines below indicate what is printed out (2 points) and give a brief explanation (3 points).

```
// ----------------------------------------

int main(int argc, char **argv) {
  Derived dr;
  Base& br = dr;
  Base* bp = new Derived;

  dr.howdy(); // 1
  br.doody(); // 2

  bp->howdy(); // 3
  bp->doody(); // 4

}
```

Problem 3

On the next few pages is code for an integer `IntSet` class that is derived from `vector<int>`. Write 4 member functions for the class.

- A new constructor that takes an array of integers and an integer count, and initializes the `IntSet` to hold the integers in the array
- A `union` member function that takes another `IntSet` and returns a new `IntSet` which has the elements of both sets.
- An `intersection` member function that takes another `IntSet` and returns a new `IntSet` which has all elements which are in both sets.
- A `difference` member function that takes another `IntSet` s and returns a new `IntSet` which contains the elements that don't appear in s

```
#ifndef _INT_SET_H_
#define _INT_SET_H_ 1

#include <iostream>
#include <vector>
using std::vector;
using namespace std;

// This class inherits from the template class
// vector<int> so it's an ordered collection of integers
// vectors are optimized for adding new elements to the end.

class IntSet : public vector<int> {
 public:
  // Create an empty set
  IntSet();

  // Initialize a set from, an array of integers
  IntSet(int* ints, int cnt);

  // Test whether i is in the set
  bool is_member(int i);

  // Add an integer to the set
  void add(int i);

  // Delete an integer from the set
  void del(int i);

  // Test whether idx is in the set
  bool operator[](int idx);

  friend ostream& operator<<(ostream& os, IntSet& is);
 private:

};

ostream& operator<<(ostream& os, IntSet& is);
#endif
```

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

#include "IntSet.h"

// Nothing really exciting here, other than using
// submember initialization
IntSet::IntSet()
  : vector<int>()
{ }

// Since I'm a vector<int> and I know that the vector<int>
// part of an IntSet has already been initialized, it's safe
// to call the push_back member function
IntSet::IntSet(int* ints, int cnt) {
    for (int i = 0; i < cnt; i++) {
      if (!is_member(ints[i])) {
        push_back(ints[i]);
      }
    }
}

// Again, an example of using inherited member functions directly
// size, and at which are defined in vector<int>
bool IntSet::is_member(int i) {
  for (int j = 0; j < size(); j++) {
    if (i == at(j)) return true;
  }
  return false;
}

void IntSet::add(int i) {
  if (!is_member(i)) {
    push_back(i);
  }
}

// vectors provide iterators to iterate sequentially
// through their elements. You can also delete elements
// from the collection using an iterator
void IntSet::del(int i) {
  vector<int>::iterator ix = begin();
  while (ix < end()) {
    if (*ix == i) {  erase(ix); return; }
    ix++;
  }
  return;
}
```

```
// This can't be used on the left hand side of an assignment
// since it doesn't return a reference type.
bool IntSet::operator[](int idx) {
  return is_member(idx);
}

// This doesn't actually have to be a friend of the
// IntSet class since it doesn't use any private members
ostream& operator<<(ostream& os, IntSet& is) {
  os << "BitSet[";
  for (int i = 0; i < is.size(); i++) {
    os << is.at(i) << " ";
  }
  os << "]";
  return os;
}
```

Problem 4

```
#ifndef _CONS_H_
#define _CONS_H_ 1

class OurString {
 public:
  OurString(char* null_string);
  OurString(char c);

  int len();
  int set_char(int index, char c);




 private:

  int buf_cnt;
  char* buf;

};
#endif
```

10 points. The above `OurString` class is intended to represent a sequence of charcters that can hold any character including `'\0'`. `buf_cnt` represents the number of characters currently in the string. `buf` is completely private storage of an `OurString` instance's characters.

Rewrite the class so that it uses the builtin `string` class in a "has-a" relationship. Change the class declaration and write implementations for all of the declared member functions.

10 points. Add two functions to the `OurString` class. A `reverse` function that returns a reversed `OurString` and an overloaded output operator for `OurString`s.

Problem 5

10 points Give two reasons, with brief explanations, why polymorphic functions are better than using switch statements in classes.

10 points Briefly explain what a pure virtual function is and why they're important. Also, give a brief code example of declaring one.

Problem 6

For each of the following statements indicate wheter it is true or false (2 points) and give a brief explanation (2 points).

By default, C++ member functions are polymorphic.

Only pointers allocated using `new` can be `deleted`

Template functions have to be member functions.

A compiler analyzes a program, builds an internal representation of the program, and immediately executes the program's actions.

In C++ we can only overload binary operators