CSC112 Fall 2010
Fundamentals of Computer Science
Solutions to Practice problems for the Final Exam

1. What is printed by the following program ?

```
// This example shows "dirty tricks" that can be done with
// pointer coercion.
#include <iostream>
using namespace std ;

int main()
{
   int a ;
   char * p ;

   a = 1 ;
   p = (char *) & a ;
   p[1] = p[0] ;     // Running on a "little-endian" machine.

   cout << "a = " << a << end ;

}
```

Answer: a = 257

2. What is printed by this one ?

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

int main()
{
   int a, b, c ;
   int *p, *q, *r  ;

   a = 17 ; b = 23 ; c = 0 ;

   p = &a ; q = &b ; r = &c ;

   *r = *q + *p ;

   (*p)++ ;

   cout << "a = " << a << "  b = " << b << "  c = " << c  << endl ;
}
```

Answer:

```
a = 18  b = 23  c = 40
```

3. Review the practice problems for exam 3.

4. Review the solutions for exam 3.

5. Recall the following class to implement a linked list.

```
class list_item {
   private:
     char * word ;
     list_item * next ;
     friend class list ; // Makes class list a friend of class list_item
} ;

class list {
   private:
      list_item * head ;
   public:
      list() { }   // Constructor does nothing.
      ~list() { }  // Destructor does nothing..
      void initialize() {  head = NULL ; }
      void print_list() ;
      list_item * find( const char * target_word ) ;
      void append( const char * new_word ) ;
} ;
```

Write the member function `append()`. The function takes a target word, and appends that word to the end of the linked list.

Answer: Here is a complete program, including `append()`.

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

class list_item {
   private:
     char * word ;
     list_item * next ;
     friend class list ; // Makes class list a friend of class list_item
} ;

class list {
   private:
      list_item * head ;
   public:
      list() { }   // Constructor does nothing.
      ~list() { }  // Destructor does nothing..
      void initialize() {  head = NULL ; }
      void print_list() ;
      list_item * find( const char * target_word ) ;
      void append( const char * new_word ) ;
```

```
} ;
void list::print_list()
{
   list_item * p ;

   p = head ;

   while ( p != NULL ) {
      cout << p->word ;
      p = p->next ;
      if ( p != NULL ) cout << " " ;
   }
   cout << endl ;
}

void list::append( const char * new_word )
{
    if ( head == NULL ) {
      head = new list_item ;
      head->word = strdup( new_word ) ;
      head->next = NULL ;
    }
    else {
       list_item * p ;
       p = head ;
       while ( p->next != NULL ) {   // p can not be NULL the first time into
          p = p->next ;              // the loop.
       }
       //  Here, p points to the last list_item object on the list.
       list_item * tail = new list_item ;
       tail->word = strdup(new_word) ;
       tail->next = NULL ;
       p->next = tail ;
    }
}

// ---------------- M A I N -----------------
int main()
{
   list L ;

   L.initialize() ;

   L.append( "cat" ) ;
   L.append( "fish" ) ;
   L.append( "dog" ) ;
   L.append( "dish" ) ;
   L.print_list() ;
}
```

6. Refer to the `job_applicant` class of the `inherit_example2` handout. Write an overloaded assignment operator that does a deep copy of objects of type `job_applicant`.

Answer: In writing and compiling an answer to this problem, I encountered a quirk of C++ that I had overlooked. When you have an object which is qualified by "const", there are constraints on calling member functions. The problem occurs when calling a non-constant function member of a constant object. Let us first look at a **very** simple example to illustrate:

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

class A {
  private:
     int m ;
  public:
     A(int k) { m = k ; }
     int get_m() { return m ; }
}  ;

int main()
{
    const A a(55)  ;

    cout << a.get_m() << endl ;
}
```

```
---------------------- Sample Session  ----------------------
atlas% g++ const_example.cc
const_example.cc: In function ‘int main()’:
const_example.cc:16: error: passing ‘const A’ as ‘this’ argument
                     of ‘int A::get_m()’ discards qualifiers
```

There are a few work-arounds to this "feature" of C++. The first one is illustrated below. We can declare the function itself to be constant. It is allowable to call a constant function member of a constant object.

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

class A {
  private:
     int m ;
  public:
     A(int k) { m = k ; }
     int get_m() const { return m ; }  // Bizarre syntax, No ?
}  ;
```

```
int main()
{
    const A a(55)  ;

    cout << a.get_m() << endl ;
}


---------------------- Sample Session  ----------------------
atlas% g++ const_example2.cc
atlas% a.out
55
```

So if we want to use the approach of declaring the needed functions to be "const", then we must modify the base class (person).

An alternate approach is to delare a new non-constant object, and coerce the constant object in an assignment to the non-constant object. This approach is taken in my solution to the overloaded assignment operator.

```
job_applicant & job_applicant::operator=( const job_applicant & a )
{
    if (&a != this) {    // I.e., not a self-copy.

       // There is a problem with calling a.get_name()
       // because of the ``const'' modifier for ``a''.
       // A clumsy, but functional work-around is to cast the
       // ``a'' as ``job_applicant''.  (Notice:  no longer ``const''.)
       //
       //  A better solution might be to use ``protected'' in the base
       //  class so that the members of
       job_applicant b ;
       b = (job_applicant )  a ;
       set_name( b.get_name() ) ;
       set_phone( b.get_phone() ) ;
       ssn = strdup( a.ssn ) ;
       jobt = a.jobt ;
    }
    return *this ;  // *this should be returned regardless of the
}                   // self-copy condition.
```

7. Vectors. Write a function to find the largest element in a vector of integers.

   Answer: Here is a complete program to illustrate vectors.

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

```
int find_largest( vector <int > & v )
{
   if ( v.size() == 0 ) {
      cerr << "find_largest():  zero length vector." << endl ;
      exit(1) ;
   }
   int largest = v[0] ;
   for ( int i = 1 ; i < v.size() ; i++ )  {
      if ( largest < v[i] ) largest = v[i] ;
   }
   return largest ;
}

int main()
{
   vector <int> a ;

   a.push_back( 7 ) ;
   a.push_back( 19 ) ;
   a.push_back( 11 ) ;
   a.push_back( 3 ) ;
   a.push_back( 8 ) ;

   cout << "largest = " << find_largest(a) << endl ;
}

----------------------- Sample Session -----------------------
atlas% g++ v.cc
atlas% a.out
largest = 19
```

8. The number of selecting k objects from a set of n distinct objects is denoted $\binom{n}{k}$. This number can be expressed by the recursive formula[1]

$$\begin{aligned}\binom{n}{k} &= 1 \qquad \text{if } k == 0 \text{ or } k == n \\ &= \binom{n-1}{k-1} + \binom{n-1}{k} \qquad \text{otherwise.}\end{aligned}$$

Write a recursive function to compute $\binom{n}{k}$ using the formula above.

```
int binomial( int n , int k )
{
    if ( ( k == 0 ) || ( k == n ) ) return 1 ;
    else  return binomial( n-1, k-1 ) + binomial( n-1, k ) ;
}
```

[1] There was an error in the base case as printed in the earlier handout. The correct recursive formula for $\binom{n}{k}$ is given here.

9. Refer to the `person` class in the `inherit_example2` handout.

   Write a derived class called `employed_person`. An `employed_person` has a job title, and an annual salary (new data members in the derived class). The job title should be represented by a character string (`char *`), and the salary represented by a numerical value (`double`). The derived class should include functions `void set_title(char * t)`, `void set_salary(double s)`, and `void print()`.

Answer: Here is a complete program including derived class "`employed_person`".

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

static char empty[1] = { '\0' } ;
//
class person {
  private:
    char * name ;
    char * phone ;
  public:
    person() ;
    ~person() ;
    void set_name( const char * aname ) ;
    char * get_name() ;
    void set_phone( const char * phone ) ;
    char * get_phone() ;
} ;

person::person() {
   name = empty ; phone = empty ;
}

person::~person() {  /* Does nothing. */   }

void person::set_name( const char * aname ) { name = strdup(aname) ; }

char * person::get_name( ) {  return name ; }

void person::set_phone( const char * aphone ) { phone = strdup(aphone) ; }

char * person::get_phone( ) {  return phone ; }

//  For this example, derived class ``job_applicant'' is omitted, but
//  it is perfectly possible to have two derived classes: ``job_applicant''
//  and ``employed_person''.
//

// -------------  Inheritance -------------
//  An object of type "employed_person" is a specialized
```

```
//  kind of person.  It inherits all attributes of a person, and
//  has some unique attributes of its own.
//

class employed_person : public person
{
    private:
      char * job_title ;
      double annual_salary ;
    public:
      employed_person() ;
      ~employed_person()  ;
      void set_title( const char * jtitle ) ;
      void set_salary( const double the_salary ) ;
      void print() ;
} ;

employed_person::employed_person()
{
    job_title = empty ;
    annual_salary = 0.0 ;
}

employed_person::~employed_person()  {  /* Does nothing */  }

void employed_person::set_title( const char * jtitle )
{
   job_title = strdup( jtitle ) ;
}

void employed_person::set_salary( double the_salary )
{
   annual_salary = the_salary ;
}

void employed_person::print( )
{
   cout << "Name: " << get_name() << endl ;
   cout << "Phone: " << get_phone() << endl ;
   cout << "Title: " << job_title << endl ;
   cout << "Salary:  $"  <<  annual_salary << endl ;
}

// ------------------------------------------------------------------
int main()
{
   employed_person P ;

   P.set_name( "Joe Worker" ) ;
   P.set_phone( "555-5555" ) ;
```

```
   P.set_title( "Senior Chief of Title Making" ) ;
   P.set_salary( 100.00 ) ;  // Pay is low for title makers.
   P.print() ;
}


---------------------- Sample session ----------------------
atlas%
atlas% g++ employed_person.cc
atlas% a.out
Name: Joe Worker
Phone: 555-5555
Title: Senior Chief of Title Making
Salary:  $100
```