CSC112 Spring 2011
Fundamentals of Computer Science
Practice Problems for the Final Exam: ANSWERS

Note: Some of these problem are too long (time-wise) to include on a 50-minute exam. Also, there are too many problems here to complete in a 50-minute exam. But, they review the material, and help prepare you for the exam.

1. Pointers and Linked lists.

   (a) What is the advantage of using a linked list instead of a fixed length array ?

   Answer:

   - A linked list can expand and contract dynamically in response to an application program's needs. A linked list uses an amount of memory proportional to the actual storage need. In contrast, a fixed length array may waste memory when only a few items need to be stored. Also, a fixed length array can not be extended at run-time if the application program's needs exceed the fixed length.
   - An item can be efficiently placed at the front of a linked. In contrast, all elements in an array must be moved forward to make a space at the front.

   (b) The following (incomplete) program uses a class to implement a linked list of cars and their gas mileage rating. Your task is to write function `best_mpg()`. Your function should return the name of the car with the best (highest) mpg. To simplify this problem, there is only one class, and the linked list itself is represented by a pointer to an `ilist_cell`. The two functions `prepend` and `best_mpg` are declared as friends, so they can have access to the private data members.

   Answer:

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

// ------------------------------------------------------
     class ilist_cell {
        private:
          char * car ;
          double mpg ;
          ilist_cell * next ;
        public:
          ilist_cell() {  car = NULL ; mpg = 0.0 ; }  // Constructor.
          friend void prepend( ilist_cell * & p, const char * acar,
                               double xmpg ) ;
          friend char * best_mpg( ilist_cell * p ) ;
     }  ;



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

```
void prepend( ilist_cell * & p, const char * acar, double xmpg )
{
   ilist_cell * t ;
   t = new ilist_cell ;

   t->car = strdup( acar ) ;
   t->mpg = xmpg ;
   t->next = p ;
   p = t ;
}

// ------------------------------------------------------------
char * best_mpg( ilist_cell * p )
{
   // Your code goes here.
   double max_mpg ;
   char * max_car ;

   max_mpg = p->mpg ;
   max_car = p->car ;

   while ( p != NULL ) {
      if ( max_mpg < p->mpg ) {
         max_mpg = p->mpg ;
         max_car = p->car ;
      }
      p = p->next ;
   }
   return max_car ;
}

// ------------------------------------------------------------
int main()
{
   ilist_cell * p ;

   p = NULL ;
   prepend( p, "corolla", 31.2 ) ;
   prepend( p, "neon", 28.4 ) ;
   prepend( p, "volt", 48.8 ) ;
   prepend( p, "miata", 26.5 ) ;

   cout << best_mpg( p ) << endl ;
}
```

Write the function `best_mpg` that returns a `char *` to the name of the car with the highest gas mileage rating.

2. Consider the following class to represent arrays of numbers:

```
class nums {
  private:
      int max ;      // Allocated length of dp.
      int n ;        // Current number of items in the array.
      double * dp ;  // Dynamically allocated array.
  public:
     // Operators defined here.
} ;
```

To properly manage memory used by objects of class `nums`, it is necessary to add: 1) a copy constructor, 2) a destructor, and 3) overloaded assignment operator.

Multiple choice:

(a) Which is true about the input to the copy constructor:

i. the input parameter is passed by value
ii. the input parameter is passed by reference
iii. there is no input parameter

Answer: the input parameter is passed by reference

(b) Which is true about the return value of the copy constructor:

i. the return value is passed back to the caller by value
ii. the return value is passed back to the caller by reference
iii. there is no return value

Answer: there is no return value

(c) Which is true about the destructor ?

i. the input parameter is passed by value
ii. the input parameter is passed by reference
iii. there is no input parameter for a destructor

Answer: there is no input parameter for a destructor

(d) Which is true about the overloaded assignment operator ?

i. the input parameter is passed by value
ii. the input parameter is passed by reference
iii. there is no input parameter

Answer: input parameter is passed by reference

(e) Which is also true about the overloaded assignment operator ?

i. the return value is returned by value
ii. the return value is returned by reference
iii. there is no return value

Answer: the return value is returned by reference

3. Refer to class `num` in question 2. Write an overloaded assignment operator for this class. Be sure to check for self-copy.

   Answer: In the public section of `class nums`, add the declaration:

```
nums & operator=( nums & rhs ) ;
```

   After the end of the class declaration, we write the implementation of `operator=`.

```
nums & nums::operator=( nums & rhs )
{
    if ( this != &rhs ) {  // Check for self-copy.
       max = rhs.max ;
       n = rhs.n ;
       dp = new double [ max ] ;
       for ( int i = 0 ; i < n ; i++ ) dp[i] = rhs.dp[i] ;
    }
    return *this ;
}
```

4. For this problem, refer to `class nums` from problem 2.

   Add an overloaded `operator+` to class `nums` Write the implementation for the overloaded `operator+` as a member function. The operation should produce a new list which numerically adds the values of the two operand lists. I.e., `a + b` should produce a new object containing the numbers from `a` added to the numbers from `b`. If the two lists are not the same length, your operator should print an error message and exit.

   Answer: In the public section of `class nums`, add the declaration:

```
nums operator+( nums & rhs ) ;
```

   Notice that most operators return their results by value, including `operator+` declared above. The overloaded assignment operator is an exception, and must return its result by reference.

   The implementation of `operator+` is shown below:

```
nums nums::operator+( nums & rhs )
{
   nums result ;

   if ( n != rhs.n ) {
      cerr << "Error: Mis-matched lengths." << endl ;
      exit(1) ;
   }
   result.max = max ;  // Take allocated length from lhs.
   result.n = n ;
   result.dp = new double [ max ] ;
   for ( int i = 0 ; i < n ; i++ ) {
       result.dp[i] = dp[i] + rhs.dp[i] ;
   }
   return result ;
}
```

5. For this problem, also refer to `class nums` from problem 2.

   Write a derived class using class `nums` as a base class. For this problem:

   (a) Make any changes necessary to class `nums` to enable the derived class to access the data members in the base class.

   Answer: In `class nums`, change "private" to "protected":

```
protected:
    int max ;      // Allocated length of dp.
    int n ;        // Current number of items in the array.
    double * dp ;  // Dynamically allocated array.
public:
   // Operators defined here.
   nums & operator=( nums & rhs ) ;
   nums operator+( nums & rhs ) ;
} ;
```

   (b) Write a member function in the derived class called `add_nums` . This function should do basically the same thing as `\operator+` in problem 4, except implement it as a function instead of an operator.

   Answer: The derived class `better_nums` is declared as follows:

```
class better_nums : public nums
{
    private:
          // Any new private data goes here.

    public:
        better_nums add_nums( better_nums & rhs ) ;

        // Other member functions go here.
} ;
```

   Function `add_nums` can be implemented as follows:

```
better_nums better_nums::add_nums( better_nums & rhs )
{
      better_nums result ;

      if ( n != rhs.n ) {
         cerr << "Error: Mis-matched lengths." << endl ;
         exit(1) ;
      }
      result.max = max ;  // Take allocated length from lhs.
      result.n = n ;
      result.dp = new double [ max ] ;
      for ( int i = 0 ; i < n ; i++ ) {
          result.dp[i] = dp[i] + rhs.dp[i] ;
      }
      return result ;
}
```

6. What is the purpose of the copy constructor and the overloaded assignment operator ?

   Answer:

   The purpose is to control the meaning of "copy" as it occurs in passing parameters, in returning parameters and in assignment statements. If we define "copy" in all of these cases to mean "deep copy", we can safely allow the destructor to free dynamically allocated memory for objects which go out of scope.

7. What is the purpose of the template class feature of C++ ?

   Answer:

   The purpose of templates is to enable code to be written once for commonly used structures, and re-used for many different underlying data types.

   For example, we have discussed the concept and implementation of a "list" data structure in several contexts (e.g., fixed length arrays, dynamically allocated arrays, and linked lists). Whenever we refer to a "list", it must be a list of some underlying type, e.g., "list of integers", or "list of names".

   Template `class vector` allows us to define and operate on lists of any user-defined data type.