Programming Design, Spring 2013
Suggested Solution for Midterm Exam

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

1. (a) False. The character type is actually a special type of integers.
   (b) False. It is only required that a `long int` must be no shorter than an `int` and a `short int` must be no longer than an `int`.
   (c) True.
   (d) True.
   (e) True.
   (f) False. The second comparison will not be performed.
   (g) False. There will be a compilation error.
   (h) False. It can be accessed only after its declaration.
   (i) True.
   (j) False. It returns the number of bytes allocated to `arr`.

2. (a) Function signatures are used to distinguish multiple functions having the same name. Because when we invoke a function, we need to know which function to invoke before we determine the return value, only the function name and parameter list are included in a function signature.
   (b) Suppose we run the insertion sort algorithm on an array. After we find the correct place to insert a value, we need to move all values after it for one position. This part cannot be improved by binary search and in the worst case requires around k steps, where k is the current length of the sorted sublist.

   Note. In the future you will learn how to implement a list with a *linked list*. In that case, no move is required. Nevertheless, you will see binary search is not doable on a linked list.
   (c) The statements are `delete [] ptr;` and `ptr = NULL;`. The computer cannot automatically release this space because it has no name to be recorded in the variable-address map.
   (d) Once we do so, all programmers in the world may use `clock_t` to be the return type of the function `clock()`. Even if in the future the data type of `clock_t` is modified in the C++ standard, as long as in `<ctime>` we still define `clock_t` with a `typedef` statement, no one needs to modify her/his own program.

3. (a) 5.
   (b) Unpredictable.
   (c) Unpredictable.
   (d) -1.
   (e) 0.
   (f) 1.
   (g) 6.
   (h) 0.

4. (a) The `else` statement in line has no `if` to be paired with.
   (b) -55 and 1045.
   (c) The program after rewritten is

```
int a;
cin >> a;
if(a == 0)
{
    for(int i = 0; i <= 10; i++)
        a += i;
    if(a > 10)
        a -= 10;
    if(a == 55)
        cout << a << endl;
    else if(a < 55)
        cout << a - 100 << endl;
    else
        cout << a + 100 << endl;
}
else
    while(a > 0)
        a -= 10;
cout << a + 1000 << endl;
```

5. (a) 16, 22 (as the average of 21 and 23), 20.5 (as the average of 20 and 21), 20.

   (b) The four lists are

   - $(2, 6, 7, 9, 12, 16, 10, 15, 8)$,
   - $(2, 6, 7, 9, 10, 12, 16, 15, 8)$,
   - $(2, 6, 7, 9, 10, 12, 15, 16, 8)$, and
   - $(2, 6, 7, 8, 9, 10, 12, 15, 16)$.

   (c) For point (x_i, y_i), we may define a "score" $s_i = 1000x_i + y_i$. As $0 \leq y_i \leq 100$, we may then safely sort these points simply based on their scores.

   (d) i. 21.

   ii. The loop that calculate `strange(6)` is

   ```
   int array[6];
   array[0] = 1;
   array[1] = 1;
   for(int i = 2; i < 6; i++)
       array[i] = array[i - 1] + 2 * array[i - 2];
   ```

   After the execution of this loop, `array[5]` contains the value `strange(6)`.

6. (a) The struct we define is

   ```
   struct Complex
   {
       double real;
       double imaginary;
       void conjugate();
       double absoluteValue();
   };
   ```

   (b) The implementation is

   ```
   void Complex::conjugate()
   {
       imaginary = -imaginary;
   }
   ```

   (c) The implementation as a global function is

```
Complex multiplication(Complex n1, Complex n2)
{
    Complex prod;
    prod.real = n1.real * n2.real - n1.imaginary * n2.imaginary;
    prod.imaginary = n1.imaginary * n2.real + n1.read * n2.imaginary;
    return prod;
}
```

This function is not designed to be a member function because it is not an operation that should be performed on a complex number.

(d) The implementation as a global function is

```
void findRoots(double a, double b, double c, Complex& root1, Complex& root2)
{
    if(b * b - 4 * a * c >= 0)
    {
        root1.real = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
        root1.imaginary = 0;
        root2.real = (-b - sqrt(b * b - 4 * a * c)) / (2 * a);
        root2.imaginary = 0;
    }
    else
    {
        root1.real = -b / (2 * a);
        root1.imaginary = sqrt(-b * b + 4 * a * c)) / (2 * a);
        root2.real = -b / (2 * a);
        root2.imaginary = -sqrt(-b * b + 4 * a * c)) / (2 * a);
    }
}
```