Test 1: CPS 6

50 Minute Exam

February 17, 1999

Name (print): ___________________________ Lecture # _____ Lab # ______

Honor Acknowledgment (signature): _______________________________________

DO NOT SPEND MORE THAN 10 MINUTES ON ANY OF THE QUESTIONS! If you do not see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 8 pages.

If you think there is a syntax error, then ask. You may assume any include statements are provided.

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>(10) pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>50 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1: (What DID you do? (12 pts))

Part A: (6 points)
Indicate the output of the cout statements below. Assume the statements are part of a program that compiles and runs. You can show your reasoning for partial credit. Note the use of reference parameters.

```cpp
int funcA (int x)
{
    x += 7;
    return x;
}

void funcB (int & y)
{
    y = 42;
}

void funcC (int z, int & w)
{
    z = 0;
    w = 0;
}

int main ()
{
    int r = 7;
    int s = 11;
    cout << r + s << " = r + s" << endl;

    r = funcA(s);
    cout << r << " " << s << endl;
    if (r > s && (r % 2 == 0))
    {
        cout << "quack" << endl;
    }

    funcB(s);
    cout << r << " " << s << endl;
    if (s > r || s / 0 == 23)
    {
        cout << "quack quack" << endl;
    }

    funcC(r, s);
    cout << r << " " << s << endl;
    if (r != s)
        cout << "quack quack quack" << endl;

    return 0;
}
```

2
**Part B:** (6 points)

Consider the function **Mystery** below.

```cpp
string Mystery (string input) {
    int len = input.length();
    input = input.Downcase();
    cout << "input = " << input << " length = " << len << endl;

    int pos = input.find("_");
    while (pos != NPOS) {
        int last = (len - 1) - pos;
        input = input.substr(0, pos) + " " + input.substr(pos + 1, last);
        cout << "input = " << input << " last = " << last << endl;
        pos = input.find("_");
    }

    cout << "input = " << input << " length = " << len << endl;
    return input;
}
```

Show the output of the **cout** statements for the following calls to **Mystery**:

- ```Mystery("Lorax");```
- ```Mystery("One_fish_tWo_FISH");```

What are pre- and post-conditions for the function **Mystery** above?
**PROBLEM 2 :**  (Form letters: (12 points))

**Part A:** (5 points)
Write the function `PrintPINLetter` whose header is given below. `PrintPINLetter` prints out a letter informing a given person of the personal identification number (PIN) for a given card. For example, the following calls:

```cpp
PrintPINLetter("Robert", "Discover", 8164);
cout << endl;
PrintPINLetter("Nisha", "MCI calling", 6621);
```

should generate the output below:

```
Hello Robert,
Your PIN for your Discover card is 8164.
Now please destroy this note.

Hello Nisha,
Your PIN for your MCI calling card is 6621.
Now please destroy this note.
```

Complete the function `PrintPINLetter` below.

```cpp
void PrintPINLetter (string name, string card, int pin)
// post: form letter to name about card is printed out
//       to terminal output: cout
{
```
Part B: (7 points)

Write the function `PrintFormLetters` whose header is given below. `PrintFormLetters` is given the name of a file containing name and card information for each member. In particular, each line of the file contains the name of the member, followed by the name of the card.

For example, suppose a data file contains the following information; then your function should generate similar (but not necessarily the exact) output to those two calls shown in Part A.

```
Robert Discover
Nisha MCI_calling
```

You may assume that each entry is correctly formatted, all names are only one word, and always in the correct order. Do not forget to close the file when you done reading the file.

To create a new random PIN for each member, use the function `MakePIN`. For this problem, you do NOT need to implement the function `MakePIN` and you may assume that it works correctly.

```
int MakePIN ()
// post: returns a number with exactly 4 digits
// each of which is randomly generated
```

You MUST use the function `PrintPINLetter` from the previous part. Assume that it works correctly even if your implementation does not.

Complete the function `PrintFormLetters` below.

```}
void PrintFormLetters(string filename)
// pre: filename is the name of a valid file (assume it can be opened)
// post: prints a form letter for each subscriber listed in fileName
{

    ifstream input;
    input.open(filename);
```
Consider the following function (lines are numbered for easier reference):

```c
1: int MakePIN ()
2: // post: returns a number with exactly 4 digits
3: // each of which is randomly generated
4: {
5:   int pin = 0;
6:   Dice d(10); // generate random digits 1..10
7:   while (d.NumRolls() < 4)
8:     { int nextDigit = d.Roll();
9:       pin = pin * 10 + nextDigit;
10:   }
11:   return pin;
12: }
```

The intent of the function is to create a number which has exactly four digits. Often it works, however, sometimes it returns a PIN with five digits. In such cases, the first digit is always one. For example, the following calls:

```c
cout << MakePIN() << endl;
cout << MakePIN() << endl;
cout << MakePIN() << endl;
```

might generate the output shown below:

```
8164
10858
6621
```

Explain what the error in the program is (i.e., why it behaves the way it does) AND how the error can be corrected.
A class `PINIterator` is designed to facilitate processing information about all PINs that have been created by a program. A `PINIterator` variable can be used to process all PINs in the order they were created once the member function `First` is called.

// class for iterating over each PIN created
// PINIterator () -- constructs an iterator
// (but iterator cannot be used until First is called)
// void First () -- starts the iterator at the first PIN
// int Current () -- returns current PIN
// e.g. 6621
// bool IsDone () -- returns true if there are no more PINs to process
// (in which case it does not make sense to call Current)
// otherwise returns false
// void Next () -- advances to the next PIN

class PINIterator
{
public:
    PINIterator (); // constructor
    void First (); // reset iterator
    bool IsDone (); // true if iterator is done
    void Next (); // advance to next PIN
    int Current (); // returns current PIN

private:
    bool CheckUnique (int pin);
    // rest not shown (and not useful)
};

Write the function `IsUnique` whose header is given below. `IsUnique` should return true if the given PIN has not already been created and false if it has. To verify if a PIN has been previously created, it checks the given pin against all PINs that have been created by the program.

Complete the function `IsUnique` below.

bool IsUnique (int pin)
// post: returns true if pin has not yet been created, false otherwise
{
    PINIterator pinIter;
PROBLEM 5: (Truly Unique: 10 points)
EXTRA CREDIT **** OPTIONAL ***** EXTRA CREDIT

Part A: (7 points)
Write the function MakeUniquePIN whose header is given below. MakeUniquePIN should return a four digit PIN that is unique. To ensure that a PIN is unique, it should keep generating PINs until one is made that has not been created before.

To create a new random PIN, use the function MakePIN. For this problem, you do not need to implement the function MakePIN and you may assume that it works correctly.

```c
int MakePIN ()
// post: returns a number with exactly 4 digits
//       each of which is randomly generated
```

To check that a PIN is unique, use the function IsUnique. You may assume this function works correctly even if your implementation in the previous problem does not.

Complete the function MakeUniquePIN below.

```c
int MakeUniquePIN ()
// post: returns a number with exactly 4 digits
//       each of which is randomly generated
//       that has not been created before
{
```

Part B: (3 points)
Explain why such a function is necessary (i.e., why is it possible to generate two identical PINs) and then discuss why you can or cannot guarantee that this function will eventually stop and return a unique PIN.