Test 1: CPS 06

Owen Astrachan       Susan Rodger

October 2, 1995

Name: _______________________________________________________

Honor code acknowledgement (signature) __________________________

<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>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 7</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Extra</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>70 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 10 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.
PROBLEM 1:  \( Output: \) 12 points

Indicate the output of each of the `cout` statements below. Assume each statement is part of a program that compiles and runs. You can show your reasoning for partial credit.

```cpp
int count = 22;
double value = 4.5;
string word = "mars";

cout << 45 + 9/value << endl;

cout << (count % 7) * 5 - 2 << endl;

cout << word + "hall" << endl;

cout << 2 * value + count/4 << endl;

cout << word.length() << endl;

cout << word < "venus" << endl;
```
PROBLEM 2:  (Heeew's A Million Dollars: 10 points)

You have been hired to write a program that will print a series of form letters to people. The two function calls below should generate the output shown.

```c
main()
{
    FormLetter("Bartolomew", "Green","100 Main St.",38);
    FormLetter("Marcia", "Brady", "22 Elm Court",42);
}
```

<table>
<thead>
<tr>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dear Bartolomew,</td>
</tr>
<tr>
<td>Now that you’re 38 years old you need to provide</td>
</tr>
<tr>
<td>for the Green family. Invest for the future, Bartolomew.</td>
</tr>
<tr>
<td>Make life at 100 Main St. happy: give us money today!</td>
</tr>
<tr>
<td>Dear Marcia,</td>
</tr>
<tr>
<td>Now that you’re 42 years old you need to provide</td>
</tr>
<tr>
<td>for the Brady family. Invest for the future, Marcia.</td>
</tr>
<tr>
<td>Make life at 22 Elm Court happy: give us money today!</td>
</tr>
</tbody>
</table>

Write the function `FormLetter`.
PROBLEM 3:  (Squash those bugs : 6 points)

Consider the following program with lines numbered (#includes are not shown). The intent of the program is to generate the output shown below.

```
<table>
<thead>
<tr>
<th>Claire Smith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter amount: 25.0</td>
</tr>
<tr>
<td>Square root function works!</td>
</tr>
</tbody>
</table>
```

5: void MakeName(string name)
6: {
7:     string newname = name + " Smith";
9: }
10:
11: main ()
12: {
13:     double amount;
14:     string name = "Claire";
15:     cout << MakeName(name) << endl;
16:
17:     cout << Enter amount;
18:     cin >> amount;
19:
20:     if (sqrt(amount) * sqrt(amount) = amount)
21:     { 
22:         cout << "Square root function works!" << endl;
23:     }
24: }

When this program is compiled, there are three errors, one on line 15, one on line 17, and one on line 20. The following compiler message appears.

```
buggy.cc: In function ‘int main()’:
buggy.cc:15: void value not ignored as it ought to be
buggy.cc:15: no match for ‘operator <<(class _IO_ostream_withassign, void)’
buggy.cc:17: ‘Enter’ undeclared (first use this function)
buggy.cc:17: (Each undeclared identifier is reported only once
buggy.cc:17: for each function it appears in.)
buggy.cc:17: parse error before ‘;’
buggy.cc:20: non-lvalue in assignment
*** Error code 1
make: Fatal error: Command failed for target ‘buggy’
```

Explain what the three compiler errors are in this program, and how they should be fixed.
**PROBLEM 4**: (Triangle Cornucopia : 10 points)

A triangle is, by definition, an *abundant* triangle if its perimeter is larger than its area. Write the function `IsAbundant` whose header is given below. `IsAbundant` returns true if the triangle with sides whose lengths are given by parameters `a`, `b`, and `c` is abundant and returns false otherwise. You can use Heron’s formula (2nd C. BC) to determine the area of a triangle. If `s` is half the perimeter, i.e.,

\[
s = \frac{1}{2}(a + b + c)
\]

then the area of a triangle is given by the formula

\[
area = \sqrt{s \cdot (s - a) \cdot (s - b) \cdot (s - c)}
\]

For example, a triangle with sides 3, 4, and 5 has perimeter 12 and area \(\sqrt{6 \cdot 3 \cdot 2 \cdot 1} = 6\), so is abundant. An example of use:

```cpp
if (IsAbundant(3, 4, 5))
{
    cout << "triangle is abundant" << endl;
}
else
{
    cout << "triangle is NOT abundant" << endl;
}
```

Complete `IsAbundant` below the following header.

```cpp
bool IsAbundant(double a, double b, double c)
// precondition: a, b, c are lengths of sides of a triangle
// postcondition: returns true if triangle is abundant, otherwise
//                 returns false
```
PROBLEM 5:  \( \text{(The Third Time is Jail Time : 10 points)} \)

In a board game, two six-sided dice are thrown to determine how many positions to move. Rolling doubles, or the same number twice, lets the player roll again. For example, rolling 6 and 3 yields a 9 position move. Rolling 2 and 2, followed by 3 and 5 yields a 12 position move \((2 + 2 + 3 + 5)\) where the player rolls again because of rolling double 2's. Rolling 3 and 3, followed by 4 and 4, followed by 1 and 6 yields a 21 position move \((3 + 3 + 4 + 4 + 1 + 6)\). There is no limit on the number of times doubles can be rolled, each time doubles are rolled the sum of the dice increases the number of positions to move and the dice are rolled again.

Write the function \( \text{TotalMoves} \) whose header is given below. \( \text{TotalMoves} \) returns the total number of positions to move by rolling two six-sided dice according to the rules described above.

\[
\text{int TotalMoves()}
\]
\[
// \text{postcondition: returns sum of rolling two six-sided dice}
\]
\[
// \text{where dice are rolled until doubles are NOT thrown}
\]

```c++
int TotalMoves()
// postcondition: returns sum of rolling two six-sided dice
// where dice are rolled until doubles are NOT thrown
```
Problem 6: (Number to rebrum: 10 points)

Note that if \texttt{num} has the value 12, then \texttt{num*10 + 3} has the value 123.

Write the function \textit{Reverse} whose header is given below. \textit{Reverse} returns a number whose digits are in the reverse order of the parameter \texttt{val}. For example:

\begin{verbatim}
cout << Reverse(1235) << endl;
cout << Reverse(3) << endl;
cout << Reverse(90120) << endl;
\end{verbatim}

should generate the output below (note that trailing zeros are lost when the digits are reversed).

\begin{verbatim}
output
5321
3
2109
\end{verbatim}

\begin{verbatim}
int Reverse(int val)
//precondition: 0 < val
//postcondition: returns number whose digits are reverse of val’s digits
\end{verbatim}
PROBLEM 7:  \((Social Summa :12\ \text{points})\)

A class \texttt{DukeStudent} is designed to facilitate writing programs to access data about students at Duke. A \texttt{DukeStudent} variable provides access to information about a student after the \texttt{Init} member function is called.

```cpp
DukeStudent student;
student.Init("213-91-1897");
if (student.Name() != "noname") // 213-91-1897 is valid SSN
{
    cout << student.Name() << endl;
    cout << "GPA = " << student.GPA() << endl;
}
generates the output below.

Pat Leiapin
GPA = 3.678
```

Part of the header file \texttt{dukestudent.h} is shown below.

```cpp
// DukeStudent class
// written 9/28/95
//
// variable/object represents a student by using the
// social security number with the Init() member function
// once initialized via Init(), information about the student
// can be accessed via member functions. The Flex account
// balanced can be changed as well using the function AddFlex()
// if not properly initialized, member functions may return
// garbage values, but Name() will return the string "no name"
// for variables that are not initialized or initialized with
// an illegal social security number
//
// DukeStudent() -- construct student object
// (NOT a valid student until Init() is called).
//
// void Init(string ssn) -- initialize to student with social security #
// specified by ssn (if invalid ssn then all
// other functions return garbage)
//
// double GPA() -- returns the GPA of a student
//
// double Flex() -- Flex balance (in dollars and cents as a double)
//
// string Name() -- returns name of student "First Last", e.g.,
// "Jane Doe", returns "no name" if NOT valid student
//
// string Login() -- returns login id, e.g., "jdoe","ola","rodger"
//
// void AddToFlex(int amount) -- add amount to Flex account
// amount can be negative to remove money
```
# Part A

A Duke student has social security number 054-53-1295. Write a code fragment to print this student’s name, login id, and Flex balance.

```cpp
class DukeStudent
{
    public:
        DukeStudent(); // constructor
        void Init(string ssn); // initialize to social security # ssn
        double GPA();     // return GPA, e.g., 3.75
        double Flex();    // return flex balance, e.g., 400.32
        string Name();    // return "first last", name of student
        string Login();   // return login id
        void AddToFlex(int amount); // add amount (can be < 0) to flex
    private:
        // can’t see this part
};
```

# Part B

A data file named `flexspend.dat` contains a social security number on the first line, followed by flex account purchases, one per line. For example, the file belows shows four purchases of $1.25, $25.75, $4.95, and $8.05

```
054-53-1295
1.25
25.75
4.95
8.05
```

You are to complete a program that, if run, will generate output as shown below:

```
Enter name of data file: flexspend.dat
name is Terry Platt
initial Flex balance = $410
after expenses balance = $370
```

Complete the program below so that it will process the data file, subtracting the purchases from a student’s Flex account. Assume that the data file exists, you do not need to check for this.
#include <iostream.h>
#include <fstream.h>    // for class ifstream
#include "dukestudent.h" // for class DukeStudent

main()
{
    DukeStudent student;
}
PROBLEM 8: *(Extra Credit: Born to Run (6 points))*

This problem is NOT required.

If a die is rolled several times, a run is a sequence of consecutive rolls showing the same number. For example, a twelve-sided die might roll 1536823991141242228. There is a run of two 5’s, a run of two 9’s and a run of four 2’s.

Write a program that simulates the rolling of a twelve-sided die one million times and that outputs the length of the longest run and what the value rolled for the run is. For the sequence above, the longest run is 4, and the value rolled for that run is 2.