

# Test 1: CPS 006L.1

Owen Astrachan

October 3, 2001

Name: \_\_\_\_\_ (1 point)

Login: \_\_\_\_\_

Honor code acknowledgment (signature) \_\_\_\_\_

	value	grade
Problem 1	25 pts.	
Problem 2	12 pts.	
Problem 3	12 pts.	
Problem 4	20 pts.	
TOTAL:	70 pts.	

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 50 minutes.

In writing code you do not need to worry about specifying the proper `#include` header files. Assume that all the header files we've discussed are included in any code you write.

**PROBLEM 1 :** (*John Kemeny's Style (25 points)*)

**Part A (5 points)**

Complete the code below that prompts the user for two numbers and prints the largest number. When executed the output should be like this.

```
enter two numbers: 15 7
largest number is 15
```

Complete the code below.

```
int a,b;
cout << "enter two numbers: ";
cin >> a >> b;
```

**Part B (5 points)**

The volume of a sphere is  $\frac{4}{3} \times \pi \times r^3$  where  $r$  is the radius. Write a function `sphereVolume` so that the call below stores in `v` the volume of a sphere of radius 11.7; you can use  $\pi = 3.14159$ .

```
double v = sphereVolume(11.7);
```

Write the header and body of `sphereVolume` below.

*continued*

### Part C (5 points)

A student writes the code below, intending it to create the string "fun fun fun fun fun" where "fun" is repeated 10 times. The code compiles and runs, but instead of "fun" repeated 10 times, a string of more than 10 "fun" words concatenated together is printed. How many times does "fun" occur in the printed string and why?

```
int main()
{
    string s = "fun";
    for(int k=0; k < 10; k++) {
        s = s + " " + s;
    }
    cout << s << endl;
}
```

### Part D (5 points)

Complete the code below so that it simulates flipping a coin 10,000 times and prints the number of (simulated) heads. A two-sided Dice object should be used for a coin, with 1 indicated heads and 2 indicating tails in the simulation.

```
#include <iostream>
#include "dice.h"

int main()
{
    int headCount = 0;

    for(
        )
    {
        if (d.Roll() == 1)
        {

        }
    }

    cout << "# heads = " << headCount << endl;

}
```

### Part E (5 points)

Complete the body of `main` below so that

1. Two dice are rolled to establish a target sum (done already)
2. The dice are rolled again, repeatedly, trying to get the same target sum. Each roll is printed.
3. The total number of rolls to obtain the target is printed.

Here's a sample run.

```
prompt> rollem
rolling for 8
5 2
6 1
6 1
6 6
3 5
total of 5 rolls to get 8 gain
```

Complete the code below.

```
#include <iostream>
#include "dice.h"

int main()
{
    Dice d1(6), d2(6);
    int target = d1.Roll() + d2.Roll();

    cout << "rolling for " << target << endl;

}
```

**PROBLEM 2 :** (*Look Before (12 points)*)

The rules for when a year is a leap year (in a leap year, February has 29 days instead of 28 and there are 366 days in the year instead of 365).

- If the year is divisible by 400, it is a leap year
- otherwise, if it's divisible by 100 it is *not* a leap year
- otherwise, if it's divisible by 4 it is a leap year
- otherwise, it is *not* a leap year

The code below gives four different functions intended to determine if a year is a leap year. **Each function compiles and runs.** For each function that *does not work as intended*, circle the function and explain why it doesn't work and how to fix it. If a function works properly, simply write *works* by the function.

```
bool leap1(int year)
// post: returns true if year is a leap year, false otherwise
{
    return (year % 400 == 0) || ( year % 4 == 0 && year % 100 != 0 );
}
```

```
bool leap2(int year)
// post: returns true if year is a leap year, false otherwise
{
    Date d(2,1,year);          // February 1
    return d.DaysIn() == 29;
}
```

```
bool leap3(int year)
// post: returns true if year is a leap year, false otherwise
{
    Date first(1,1,year);
    Date last(12,31,year);

    return last-first == 365;
}
```

```
bool leap4(int year)
// post: returns true if year is a leap year, false otherwise
{
    if (year % 400 == 0)      return true;    // divisible by 400
    else if (year % 100 == 0) return false;   // divisible by 100
    else if (year % 4 == 0)  return true;    // divisible by 4
    else                      return true;
}
```

**PROBLEM 3 : (A Comedy of Errors (12 points))**

The complete program shown below (with 28 lines) is intended to prompt the user for the amount of a restaurant bill and a tip percentage and to print how much money is needed to pay the bill with the tip. The intended interaction with the program is something like this:

```
what is the bill? 40.00
What is percent tip (integer)? 10
you should pay $44.00
```

Unfortunately, the program has two syntactic/compiler errors and one logic error. The compiler errors are given below, explain why each one occurs and how to fix the problem. You can show in the code how to fix the problem or explain in words.

**Part A/Compiler Error I**

```
tipper.cpp: In function 'double withTip(double, int)':
tipper.cpp:13: warning: control reaches end of non-void function 'withTip(double, int)'
```

**Part B/Compiler Error II**

```
tipper.cpp: In function 'int main()':
tipper.cpp:9: too few arguments to function 'double withTip(double, int)'
tipper.cpp:25: at this point in file
```

**Part C/Logic Error**

If the two compiler errors are fixed properly the program will print 40.00*instead of*44.00 as the intended dialog shows above because of a logic/semantic bug. No matter what percent tip the user enters, the total to pay that's printed is always the exact same value as the value the user enters for `payThis`.

Explain why this happens and how to fix it.

*code on next page*

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. double withTip(double bill, int percent)
6. // post: return how much to pay for bill if tipping
7. //       at percent rate, e.g., if percent is 15
8. //       we're tipping 15%
9. {
10.     double tip = bill * (percent/100);
11.     double total = bill + tip;
12.     cout << total << endl;
13. }
14.
15. int main()
16. {
17.     double payThis;
18.     int tipRate;
19.     cout << "what is the bill? ";
20.     cin >> payThis;
21.
22.     cout << "what is percent tip (integer)? ";
23.     cin >> tipRate;
24.
25.     cout << "you should pay $" << withTip(tipRate) << endl;
26.
27.     return 0;
28. }
```

**PROBLEM 4 :** (*Banana Basics (20 points)*)

In this problem you'll use the function `splitOn` shown below. The questions for this problem follow the code. You should probably look at the questions before studying the code in detail.

For example, the code that follows, in which `splitOn` is called, prints these two lines of output:

```
world : hello
bad wolf : big
```

Here's the code that prints the two lines above.

```
string hello = "hello-world";
string wolf  = "big bad wolf";
string first;

if (splitOn(hello, first, "-"))
{
    cout << hello << " : " << first << endl;
}
if (splitOn(hello, first, "-"))
{
    cout << hello << " : " << first << endl;
}
if (splitOn(wolf, first, " "))
{
    cout << wolf << " : " << first << endl;
}
}
```

Here's the body of the function `splitOn`.

```
bool splitOn(string& original, string& firstPart,
             const string& separator)
// post: if original contains separator, then returns true
//       and stores substring of original from index 0
//       up to (not including) separator in firstPart,
//       and stores substring after separator in original
//       so original changes as a result of calling splitOn
//
//       else (separator not in original) returns false
//       and doesn't change original or firstPart
{
    int index = original.find(separator);
    if (index != string::npos)
    {
        firstPart = original.substr(0, index);
        original = original.substr(index+1, original.length());
        return true;
    }
    return false;
}
```

### Part A (6 points)

Briefly, in `splitOn` why is `original` a reference parameter rather than a value or const-reference parameter?

Briefly, in `splitOn` why is `separator` a const-reference parameter rather than a value or reference parameter?

### Part B (6 points)

Given the code/definitions below, fill in the call to `splitOn` so that the code prints

```
them elves
```

Here's the code, you can only fill in parameters to `splitOn`, you can make no other changes to the code.

```
string s = "themselves";
string t;
if (splitOn(           ,           ,           ))
{
    cout << t << " " << s << endl;
}
```

### Part C (5 points)

Complete the function `lastSegment` so that it returns the last segment in an internet IP address, e.g., so that the call below stores "117" in the string `last`.

```
string last = lastSegment("152.3.140.117");
```

In writing `lastSegment` you may call any `string` member functions and you may call the function `splitOn` shown earlier. Assume the precondition of `lastSegment` is true (you don't need to check for invalid ip strings as parameters).

```
string lastSegment(const string& ip)
// pre: ip contains three dots, e.g., is valid IP address like "152.3.140.117"
// post: returns the last segment (string after third dot ".") of ip
```

### Part D (9 points)

Complete function `parse` below so that it breaks a string whose parts are separated by ampersands `&` and prints the parts. For example, the call

```
parse("apple=fruit&cheese=dairy&bean=legume&cherry=fruit");
```

Should generate the following output.

```
0.    apple=fruit
1.    cheese=dairy
2.    bean=legume
3.    cherry=fruit
```

In writing `parse` you may call any `string` member functions and you may call the function `splitOn` shown earlier.

```
void parse(const string& s)
// pre: s consists of pieces separated by ampersands &
// post: prints each piece on a line by itself, with
//       each piece numbered starting at 0/zero.
{
    string original = s;    // s is const, we can change original
    string first;
    int pieceCount = 0;

    while (
    {

        pieceCount++;

    }

}
```

## String member functions

```
int string::find(const string& s) const;
// post: return position/index of first location of s
//       returns string::npos if not found

int string::rfind(const string& s) const;
// post: return position/index of last location of s
//       (reverse find, start at end of string)
//       returns string::npos if not found

string string::substr(int index, int n) const;
// post: return substring of n characters starting at index
//       ok if n too big, problems if index too big

int string::length()
// post: returns # characters in string
```

## Dice member functions

```
class Dice
{
public:
    Dice(int sides);           // constructor
    int Roll();               // return the random roll
    int NumSides() const;     // how many sides this die has
    int NumRolls() const;     // # times this die rolled
};
```

## Date member functions

```
class Date
{
public:
    Date();                   // construct date with default value
    Date(long days);          // construct date from absolute #
    Date(int m,int d,int y);  // construct date with specified values

    int Month() const;        // return month corresponding to date
    int Day() const;          // return day corresponding to date
    int Year() const;         // return year corresponding to date
    int DaysIn() const;       // return # of days in month
    string DayName() const;   // "Monday", "Tuesday", ... or "Sunday"
    string MonthName() const; // "January","February",... or "December"
    long Absolute() const;    // number of days since 1 A.D. for date
    string ToString() const;  // returns string for date in ascii

    Date operator ++(int);     // add one day, postfix operator
    Date operator --(int);     // subtract one day, postfix operator
    Date& operator +=(long dx); // add dx, e.g., jan 1 + 31 = feb 1
    Date& operator -=(long dx); // subtract dx, e.g., jan 1 - 1 = dec 31
    // more here
};
```