

**Final Exam, 3/15/09  
PIC 10A, Winter 2009**

**KEY**

**Check your TA's name:**

\_\_\_\_\_ **Vlad**

\_\_\_\_\_ **Darren**

*You have 3 hours to complete this exam. You are not allowed to use any books, notes, calculators, or electronic devices. Write your answers carefully. Incomplete, unintelligible, or illegible answers will receive little or no credit. When you are asked to write a program, it is not necessary to comment your code but you are expected to indent appropriately to make your code easier to follow. There are a total of 100 points on this exam.*

| <b>PAGE</b>  | <b>SCORE</b> | <b>POSSIBLE</b> |
|--------------|--------------|-----------------|
| <b>1</b>     |              | <b>8</b>        |
| <b>2</b>     |              | <b>8</b>        |
| <b>3</b>     |              | <b>12</b>       |
| <b>4</b>     |              | <b>6</b>        |
| <b>5</b>     |              | <b>8</b>        |
| <b>6</b>     |              | <b>8</b>        |
| <b>7</b>     |              | <b>10</b>       |
| <b>8</b>     |              | <b>18</b>       |
| <b>9-10</b>  |              | <b>22</b>       |
| <b>TOTAL</b> |              | <b>100</b>      |

1.) [2 points] In lecture we discussed that we can repeatedly read in numbers from the user using the loop

```
while (cin >> x)
```

where  $x$  is a number valued data type like `int` or `double`. Explain why this loop would not be appropriate for reading in a list of words when  $x$  is a string.

Since every possible input counts as a string, this would create an infinite loop.

2.) [3 points] Write a `for` loop that prints the characters in a string `s` one at a time, separated by a space. For example, `s="FRODO"` prints: F R O D O

```
string s;
cin >> s;

for (int i=0; i<s.length(); i++)
    cout << s[i] << " ";
```

3.) [3 points] Write a `while` loop that prints the numbers from an integer `N` down to zero, separated by commas. For example, `N=4` prints: 4, 3, 2, 1, 0,

```
int N;
cin >> N;

while ( N >= 0) {
    cout << N << ", ";
    N--;
}
```

2.) [8 points] Correct the errors in the following blocks of code. You will get points off for changing code that was not an error.

a.) 

```
string a;
cin >> a;
if (a[0] == "Q")
    cout << "Your word started with Q.";
```

Should be single quotes 'Q' on the char.

b.) 

```
string b = "Gandalf";
cout << b.substr(4,6);
b.insert(0, "Hello ");
cout << b;
```

The substr parameter 6 runs off the end of the string.  
Can be at most substr(4,3)

c.) 

```
string c = "Hello";
string* p = c;
cout << *p;
```

p is a pointer, so it needs an address not a char value  
string\* p = &c;

d.) 

```
int N;
cout << "Enter array size: ";
cin >> N;
int d[N];
for (int i=0; i < N; i++) {
    cout << "Enter number: ";
    cin >> d[i];
}
```

Array size must be const, cannot be input by user.  
Use a vector instead.  
vector<int> d(N);

**3.) [12 points]** Write the output of the following blocks of code.

**a.)**

```
for (int i=1; i<=3; i++) {
    for (int j=1; j<i; j++)
        cout << i << " ";
    cout << "\n";
}
```

2  
3 3

**b.)**

```
vector<char> b(3);
b[0]='r';    b[1]='a';    b[2]='c';
b.push_back('e');
for (int i=2; i>=0; i--)
    b.push_back(b[i]);
for (int=0; i<b.size(); b++)
    cout << b[i];
```

racecar

**c.)**

```
int* c = new int;
*c = 10;
int* p = c;
*p += 3;
cout << *c << *p;
```

1313

**d.)**

```
/**Indicate the spaces in your answer.***/
vector<string> d(3);
d[0]="Sam";  d[1]="Frodo";  d[2]="Gandalf";
int j = 2;
do {
    cout << setw(d[j].length()+2) << d[j];
    j--;
} while (j>0);
```

xxGandalfxxFrodo      where x indicates a space

4.) [6 points] Write the output of the following program. Track your variables in a walkthrough table as we did in lecture.

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

void fun(string*& p, string* q) {
    *q = "frodo";           (2)
    q--;
    *q += *(q+1);           (3)
    p = p+2;
    (*p).insert(1, "AAA");   (4)
    return;
}

int main() {
    string s[4] = {"abc", "123", "hi", "yo"}; (1)
    string *p = s;
    fun(p, &s[1]);
    for (int i=0; i<4; i++)
        cout << s[i] << " ";
    cout << "\n" << *p;
    return 0;
}

(1) s = abc 123 hi yo
(2) s = abc frodo hi yo      p->abc    q->frodo
(3) s = abcfrodo frodo hi yo p->abc    q->abcfrodo
(4) s = abcfrodo frodo hAAAi yo p->hAAAi q->abcfrodo
```

### Output

```
abcfrodo frodo hAAAi yo
hAAAi
```

5.) [8 points] Write a function **isSubstring** that takes two strings as input and returns true if the second string is contained in the first string. It should return false if the second string is not contained in the first string. Note that if the second string is longer than the first string, you can immediately return false. An example is shown below.

|                                         |               |
|-----------------------------------------|---------------|
| isSubstring("Gandalf", "dal")           | returns true  |
| isSubstring("Gandalf", "doh")           | returns false |
| isSubstring("Gandalf", "Gandalf Rocks") | returns false |

```
bool isSubstring (string s1, string s2) {  
    if (s2.length() > s1.length())  
        return false;  
    for (int i=0; i <= s1.length()-s2.length(); i++)  
        if (s1.substr(i,s2.length()) == s2)  
            return true;  
    return false;  
}
```

6.) [8 points] The median of a sorted list of numbers is the value in the middle. If the length of the list is an odd number, the median is the number in the center position. If the length of the list is even, the median is the average of the two numbers in the middle. Write a function called **median** that takes a vector of integers as input and returns the median value. You may assume the input vector is sorted from smallest to largest. An example is shown below for two lists, one of odd length and one of even length.

|                       |   |   |    |    |    |
|-----------------------|---|---|----|----|----|
| v=                    | 2 | 7 | 10 | 15 | 22 |
| median(v) returns 10  |   |   |    |    |    |
| w=                    | 3 | 8 | 9  | 15 |    |
| median(w) returns 8.5 |   |   |    |    |    |

```
double median (const vector<int>& v) {
    int N = v.size();
    if (N%2 == 0)
        return (double)(v[N/2]+v[1+N/2])/2;
    else
        return v[1+N/2];
}
```

**8.) [10 points]** We wish to create a **Dice** class that stores the values on two N-sided dice. Similar to our Card class, we want the constructor to create two random dice values. So creating an object with the statement

```
Dice my_dice(8);
```

should create two dice with values in the range [1,8]. The declaration for the Dice class is shown below. (*Vocabulary Note: The word "die" is singular for "dice".*)

```
class Dice {
public:
    Dice(int N);
    /** Possibly more functions ****/
private:
    int die1;
    int die2;
};
```

**a.)** Write the definition of the constructor function, as it would appear in the file dice.cpp.

```
Dice::Dice(int N) {
    die1 = 1+rand()%N;
    die2 = 1+rand()%N;
}
```

**b.)** We want to compare two Dice objects based on the total sum on the dice. For example, the dice {2,1} would be less than the dice {1,5} because the sum is less. Write the definition of the member function that overloads the < comparison as used below:

```
if (my_dice < your_dice)
```

```
bool Dice::operator< (Dice dice2) const {
    return die1+die2 < dice2.die1 + dice2.die2;
}
```

9.) [18 points] Sales at Hewlett-Packard are recorded in the text file "sales.txt". Each line of the file specifies a sale with the salesperson's last name and the dollar amount of the sale. The file is sorted alphabetically by name, so that each person's sales appear consecutively. Write a full program (starting from #include) that reads in the file "sales.txt" and prints to the console the total sales amount for each salesperson. An example is shown below. Note the first total is shown as \$21.50, not \$21.5.

| <b>sales.txt</b> |       | <b>Console Output</b>   |
|------------------|-------|-------------------------|
| Baggins          | 11.50 | Baggins earned \$21.50  |
| Baggins          | 10.00 | Gamgee earned \$8.90    |
| Gamgee           | 4.10  | Greenleaf earned \$4.88 |
| Gamgee           | 3.80  | Smith earned \$6.00     |
| Gamgee           | 1.00  |                         |
| Greenleaf        | 4.88  |                         |
| Smith            | 1.00  |                         |
| Smith            | 2.00  |                         |
| Smith            | 3.00  |                         |

Note: The basic strategy is to read in one name and amount at a time. If the name is the same as the previous name, add amount to the total. If the name you just read is different, output the total. Note the first iteration works because initially current\_name is a blank string. The tricky part is to make sure you output the last person in the file (Smith in the example).

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

int main() {
    ifstream fin;
    fin.open("sales.txt");
    string input_name;
    double input_amount;
    string current_name;
    double total = 0;
    cout << setprecision(2) << fixed;
    while (fin >> input_name >> input_amount) {
        if (input_name == current_name)
            total += input_amount;
        else { //New name detected.
            cout << current_name
                << " earned $"
                << total
                << "\n";
            current_name = input_name;
            total = 0;
        }
    }
    cout << current_name << " earned $" << total << "\n";
    fin.close();
    return 0;
}
```

**10.) [22 points]** Records for salespeople at Hewlett-Packard are stored in a **Salesman** class. The class keeps track of the person's name and all sales that person made. Each sale tracks the customer name and the dollar amount of the sale. An example usage of the class is shown below.

```
Salesman s("Frodo Baggins");
s.add_sale("Microsoft", 33.00);
s.add_sale("Hobbits R Us", 88.22);
s.add_sale("Orcs Inc.", 25.90);
cout << "The sales for " << s.get_name() << " are:\n";
s.print_sales();
cout << "\nTotal sales = $" << s.get_total();
```

### **OUTPUT**

```
The sales for Frodo Baggins are:
Microsoft      $33.00
Hobbits R Us   $88.22
Orc Inc.       $25.90

Total sales = $147.12
```

The `print_sales()` function should print all sales on the console screen, each on a separate line. The sales should appear formatted correctly in two columns, where the first column adapts to the length of the customer names and all numbers in the second column are lined up at the dollar sign \$.

Write the class declaration and all member function definitions for the **Salesman** class. You will have to figure out what functions to add based on their usage in the example. You may assume the `<vector>`, `<string>`, and `<iomanip>` libraries are available.

*Hint: Recall you can change setw to be left-aligned with the command: `cout << setiosflags(ios::left);`*

```
class Salesman {
public:
    Salesman(string new_name);
    void add_sale(string client, double amount);
    string get_name() const;
    void print_sales() const;
    double get_total() const;
private:
    string name;
    vector<string> clients;
    vector<double> amounts;
};
```

```

Salesman::Salesman(string new_name) {
    name = new_name;
}

void Salesman::add_sale(string client, double amount) {
    clients.push_back(client);
    amounts.push_back(amount);
    return;
}

string Salesman::get_name() const {
    return name;
}

void Salesman::print_sales() const {
    //First find max string length.
    int max = 0;
    for (int i=0; i<clients.size(); i++)
        if (clients[i].length() > max)
            max = clients[i].length();
    cout << setiosflags(ios::left);
    cout << setprecision(2) << fixed;
    //Output clients and amounts in columns.
    //I used buffer of 3 spaces. Could be different.
    for (int j=0; j<clients.size(); j++) {
        cout << setw(max+3) << clients[j]
            << "$" << amounts[j] << "\n";
    }
    return;
}

double Salesman::get_total() const {
    double total = 0;
    for (int i=0; i<amounts.size(); i++)
        total += amounts[i];
    return total;
}

```