

Exam 2

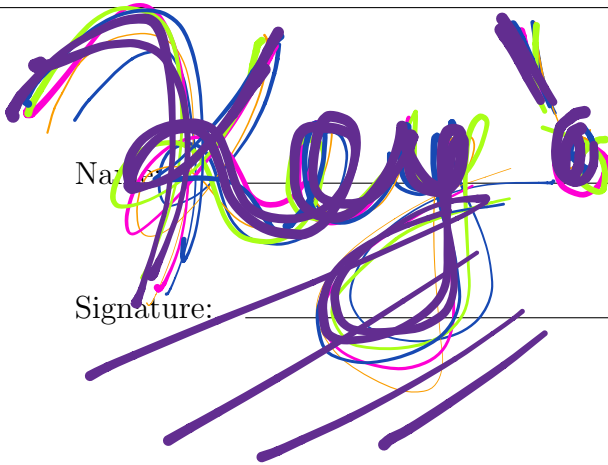
CSI 201: Computer Science 1
Fall 2017

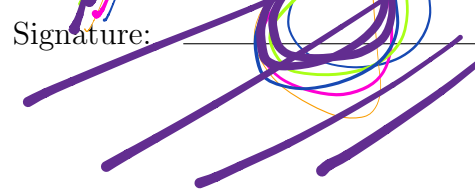
Professor: Shaun Ramsey, Ph.D.

It matters!

I understand that this exam is closed books and closed notes and is to be completed without a calculator, phone, or other computer. I am **NOT** allowed to use any external resources to complete this exam. The work that I am submitting and that I have viewed during this exam is mine. I understand that images, video and sound may be taken and recorded during this exam. I have completed this exam in accordance with the Washington College Honor Code.

For full credit, remember to use good style and programming practice throughout.

Name: 

Signature: 

CS IS
SUPER
FUN!

o Answers

o Additional Explanation

o Extra Commentary!

-For example, there are some style typos
denoted with "TAB →" indicating a tab indent should appear

2. Concepts: Answer the following briefly. When code is requested, your response should consist of less than 2-3 lines of code.

(a) 6 points What does fun1 return in the following function calls:

fun1(33.0);

true

fun1(0.0);

false

fun1(-33.0);

true

```
//f1 is given by this function
bool fun1(double funny) {
    if (funny != 0) { //truly funny
        return true;
    }
    else { //not so funny
        return false;
    }
}
```

Although comparing doubles (floats) is a "sketchy" proposition that should be done in ranges.

$\text{if } (\text{fabs}(\text{funny}) < 1e-6)$
 compares funny to
 see if it is "close"
 to 0.

(b) 2 points Name one situation in which we use call by reference and explain why.

Input functions with more than one input to allow the inputs to be changed

(c) 2 points What happens when an invalid index is accessed in a vector?

*An out of bounds error appears (crashing the program if using .at)
 If using [] as the accessor then results are unpredictable/unknown.*

- (d) 6 points What does squareMax return when the following are called?

```
squareMax(4, 3);
```

3

```
squareMax(3, 11);
```

9

```
squareMax(3, 9);
```

9

```
double squareMax(double var, double limits) {
    double value = var * var;
    TAB ← if (value > limits) {
            value = limits;
        }
    return value;
}
```

- (e) 4 points Examine the function squareMax from above. If we pass 9 as the second parameter, for what values of the first parameter will the function return 9? This is extremely useful, for example, if we wanted to write a unit test in a for loop to test a variety of values for the first parameter. In that situation, we might be able to assert that the function returns 9 in these situations. Give the full range of possible values for full credit. List a handful of values for partial credit.

{ (3, 9) from (d) works,
 (2, 9) does not
 (4, 9) does. (5, 9) does, (6, 9), ...

For all values greater than or equal to 3 or less than or equal to -3

- (f) 2 points Show a function call of the function squareMax above that returns a value of 13.

squareMax(10, 13);

The choice here is any value big enough to give more than 13 when squared. So anything 4 and above is great. Really any value > 3.6 is fine but there's no reason to use this value.

- (g) 4 points Demonstrate how to read in (from the console) a vector of 3000 strings named **names**.

<pre style="font-family: monospace; color: magenta;">for (int i = 0; i < 3000; ++i) { string s; cin >> s; names.push_back(s); } ↑</pre>	<pre style="font-family: monospace; color: magenta;">for (unsigned i = 0; i < names.size(); ++i) { string s; cin >> s; names.at(i) = s; } ↑</pre>
--	--

In this solution names is assumed to be created with no size
Ex: `vector<string> names;`

In this solution names is assumed to be created with space reserve for the 3000 strings
Ex: `vector<string> names(3000);`

Always list your assumptions

- (h) 4 points Demonstrate how to output (to the console) every other element of a vector of doubles named **grades**.

Change this 0 to a 1 to output the items in "odd" index positions instead of "even" ones ↓

```
for (unsigned i = 0; i < grades.size(); i = i + 2) {
    cout << grades.at(i) << endl;
}
}
```

Skips every other. Other options involve using %

- (i) 2 points Describe the run-time error with the following code snippet.

```
vector<int> pumpkins(42); ← creates a vector with 42 entries
for (unsigned i = 1; i <= pumpkins.size(); i++) { ← loop starts w/ i=1
    pumpkins.at(i) = 42 - i;
}
}
```

Error: When $i=42$, index is out of bounds.

and runs until $i=43$, it enters the loop with $i=42$ but not with $i=43$

3. 10 points In this problem, you have been given a vector of integers that represent ages and is aptly named **ages**. You are required to determine how many elements of the vector fall between the values of 17 and 22 (inclusive). Output (to the console) this amount as well as the total number of elements that are in the vector.

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

```
int main() {
```

```
    // assume the vector of integers named ages is defined here and
```

```
    // has been filled with all the data required
```

```
    ... ← ages is made & filled by someone else
```

```
    // Your code goes here
```

```
    int count = 0;
```

```
    for (int i = 0; i < ages.size(); ++i) {
```

```
        if (ages.at(i) >= 17 && ages.at(i) <= 22) {
```

```
            count++;
```

```
        }
```

```
    }
```

```
    cout << count << " out of " << ages.size()
```

```
    << " were between 17 and 22" << endl;
```

```
    // end of your code
```

```
    return 0;
```

```
}
```

Handwritten notes:

- 1) Make some count
- 2) For all elements
- 3) — check if b/n 17 & 22
- 4) ————— Increase count
- 5) Output count & total #

Diagram: A blue bracket on the left side of the list above groups items 2, 3, and 4. To the left of this bracket, the text "2 pts each" is written in blue, with a blue arrow pointing from the text to the bracket.

4. 10 points Compound interest helps us understand how much money can be earned (or lost) by saving (or taking a loan). To compute how much money we will have (or owe) after a certain number of years (t) at a given rate (r) when we put in (or take out) a certain amount of dollars (P), uses the equation: $A = P(1 + r)^t$. We've been asked to write a function in which P , r and t are given to the function. In return, the function computes and returns A . The `cmath` library provides the `pow(x,y)` function. `pow(x,y)` can be used to compute x^y .

```
#include <iostream>
#include <assert>
#include <cmath>
using namespace std;

double futureValue(double P, double r, double t);

int main() { //some unit tests
    assert( futureValue(1000, 0.0, 10) == 1000);
    assert( futureValue(1000, 0.1, 1) == 1100);
    assert( futureValue(1000, 1.0, 3) == 8000);
    assert( futureValue(1000, .01, 10) == 1104.52 );
    return 0;
}
double futureValue(double P, double r, double t) {
\\your code begins here

    return P * pow(1+r, t);

\\your code ends here
}
```

Making intermediate variables would also be fine here. It was more common in the answers!

Points:

- a) Using `pow` correctly
- b) proper return
- c) proper calculation
- d) proper usage of variables
- e) style of function "pieces"

Roughly 2 pts each.

Question	Points	Score
1	20	
2	32	
3	10	
4	10	
Total:	72	