Exam 1 Review - Fall 2020

# Exam Details

* The exam is made up of: writing code to solve problems.
* There are multiple versions of the exam.
* **Partial credit is available** for every question.
  + Your goal should be to not need it.
  + Focus your solution on correctness of the problem solving method. Clear evidence of your thought process and plan must be included in the code and comments in order to qualify for partial credit.
  + Put down anything you do know even if you can’t solve the entire problem. Some partial credit is better than none.
  + Make your thinking visible so applying partial credit is easier.
* **You may not consult other people** while taking the exam.
* **Open book/Open notes.**
  + You may use
    - zyBook,
    - any code you’ve written,
    - class materials including slides,
    - videos,
    - example code,
    - C++ references
      * [C++ Reference](http://www.cplusplus.com/reference/)
      * [CPP Reference](http://www.cppreference.com/)
* You may compile code and run it.
  + *Beware of getting caught up in debugging since that can quickly consume your time and result in not getting to all questions.*
* **WARNING!**
  + With open book exams, do not assume you don’t need to study...
    - You don’t want to use your time looking for answers or learning material.
    - You should know the material well enough to quickly find any supporting information you might need for quick clarification and verification.

# 

# Exam Strategies

* Understanding at a high level will help you code better during an exam. So, as you study, ask yourself…
  + *What is happening?*
    - *High level: What is the goal?*
    - *Code level: What are the effects and consequences of statements, etc.?*
  + *Why is it that way?*
    - *High level: Why are certain design decisions made? Why is it designed that way?*
    - *Code level: What is the code that accomplishes a higher-level goal? Why is it done that way?*
  + *How is it done?*
    - *High level: How do you accomplish a high-level goal?*
    - *Code level: How do you write the code?*
* Study Strategy: Prioritize how you study to maximize your ability to demonstrate concept mastery while not overwhelming yourself.
  + For a solid passing grade focus on class topics, slides, examples, homework and labwork.
    - Make sure you understand anything coded in homework, labwork, and in-class examples.
      * What general (higher level) principles were illustrated in a labwork/homework?
        + Could you explain it in general?  
          (i.e. what/why rather than how)
        + Be able to solve and write code for similar problems.
  + After you’ve done all you can do in the “for a solid passing grade” above, then focus on zyBook Challenge Activities and Participation Activities.
    - If a challenge or activity seems fuzzy, review the text for clarification.
    - Key terms are bolded.
  + After you’ve done all you can from the previous approaches and if you are still fuzzy about a topic, read about it in a different source or textbook. You can also talk to someone who can explain it in a different way.

# 

# Exam Topics

## *Note, topics are not always organized sequentially!* Getting Started

* Compile Process
* Program Flow / Control Structures
* First Program
  + Basic IO
  + Comments & Whitespace
  + Errors and Warnings

## Variables / Assignments

* Operators
* Assignment
* Expressions

## Overarching Themes

* Software Development Process
  + Design
    - Flowcharts
    - Pseudocode
* Header files
* Coding Style / Readability
* Debugging

## Selection

* Selection (e.g. if, switch)
* Logical Operators
* Boolean data type
* Conditional Expressions
* Floating-point comparison

## Iteration

* Iteration (e.g. while, for, do while)
* Nested Loops

## 

## 

## Strings

* Comparisons
* Access
* Modify
* Character Operations

## Compound Data

* Array and its use
* Parallel Arrays
  + (i.e. Multiple Arrays in zyBook)
* Linear search

## Exceptions

* Why do we use them?
* throw
* try
* catch

## Functions

* Signature/parameters/arguments (formal vs. actual)
* Call and return
* Scope
* Declarations vs definitions
* Unit Testing

## Streams

* ostream
* istream
* String streams
* Stream states
  + goodbit, eofbit, failbit, badbit
  + stream state functions
    - good()
    - eof()
    - fail()
    - bad()
* getline() vs. input extractor >>
* File streams
* File processing

# Practice Problems

Note that practice problems are generally greater than or equal in difficulty to what you will find on the exam. Exam questions are designed to be challenging and doable in the time provided. Some of the practice questions are longer than what you would see on an exam, but should give you good practice.

Additional practice problems can be doing the parts of the labworks that you did not work on.

## Multiples

Given positive integers n, k and l, print the first n positive integer numbers that are multiple of k, l or both. Example: n = 6, k = 2 and l = 3, you should print:

2 3 4 6 8 9

## Triangular numbers

A positive integer number n is triangular if it can be obtained by the product of three consecutive positive integers. For example, 120 is triangular, since 4 \* 5 \* 6 = 120. Given n > 0, determine if n is triangular.

## Array Segments

Given n > 0 and a sequence of n integer numbers, print how many segments composed by consecutive copies of the same number the sequence has.

For example: The sequence

5 2 2 3 4 4 4 4 1 1

has 5 segments.

## Find the sum

Write a program that given:

* n such as 0 < n < 5000
* a list of n numbers in increasing order
* a target number k

prints two distinct elements from the list such that their sum is equal to the target k if such pair exists and ”none” otherwise.

Examples:

* For n = 3, list 2 4 6 and k = 13, the output is none.
* For n = 8, list 1 2 3 4 5 6 7 8 and k = 8, valid outputs are:
  + 1 7
  + 2 6
  + 3 5

## 

## Numeric Palindrome

Write a boolean function that takes an integer parameter and returns true if the base 10 number is a numeric palindrome. Note, you cannot use any type of string to solve this problem.

bool isNumericPalindrome(int n)

For example

* isNumericPalindrome(121) → true
* isNumericPalindrome(-121) → true
* isNumericPalindrome(220) → false
* isNumericPalindrome(-220) → false
* isNumericPalindrome(18344381) → true
* isNumericPalindrome(12345) → false
* isNumericPalindrome(1) → true
* isNumericPalindrome(0) → true

## Happy numbers

A happy number is a non-negative integer that eventually becomes 1 when iterated over the sum of squared digits function.

For example, 28 is happy:

28 **→** 22 + 82 = 68 **→** 62 + 82 = 100 **→** 12 + 02 + 02 = 1 **✓**

But 4 is unhappy (omitting intermediate results), as the chain of numbers led to the

original number.

4 **→** 16 **→** 37 **→** 58 **→** 89 **→** 145 **→** 42 **→** 20 **→** 4 ...

In fact, every unhappy number eventually converges with 4.

Write a function bool isHappy(int n) that receives as an argument a non-negative integer and returns true if the number is happy.

Examples:

* isHappy(4) returns false.
* isHappy(13) returns true.
* isHappy(28) returns true.

Continued on the next page.

## Pesky Bug

Your friend has written a function that looks to see if there is a failing grade in a list of grades in an array. What is wrong with your friend's code?

A failing grade is anything less than 60.

#include <iostream>

#include <iomanip>

#include <stdexcept>

using std::invalid\_argument;

using std::cout, std::endl;

using std::boolalpha;

bool hasFailingGrade(double grades[], unsigned int size) {

if (size == 0) {

throw invalid\_argument("There are no grades.");

}

bool hasFailing = false;

for (unsigned int i = 0; i < size; ++i) {

if (grades[i] < 60) {

hasFailing = true;

}

else {

hasFailing = false;

}

}

return hasFailing;

}

int main() {

double g1[] = {83, 91, 100, 60, 75};

double g2[] = {100, 100, 100, 59.9};

cout << "g1 has a failing grade is " << boolalpha;

cout << hasFailingGrade(g1, 5) << endl;

cout << "g2 has a failing grade is " << boolalpha;

cout << hasFailingGrade(g2, 4) << endl;

}

## Continued on the next page.

## Has Prohibited Items

You are a landlord. As tenants move in, they send you a file with a list of what they plan to put in their apartment. There are some items that are prohibited.

Write a program that reads in a list of prohibited items and a list of tenants with their name and items they plan to put into their apartment. The program checks to see if the tenants plan to have any prohibited items in their apartment and outputs which tenants items are safe and which are not.

* Assume there are no spaces in names or items. So a name won’t be “Michael Moore” and an item won't be “gold watch”.
* Recall that >> will stop reading a string when it encounters white space.

### Input Files:

|  |  |
| --- | --- |
| prohibited.txt | tenants.txt |
| gun flamethrower  gasheater | Taylor futon chair armoire shoerack bed  Brooklyn bed couch gasheater dresser chair table Avery flamethrower marshmallow grahamcracker chocolate |

### Output File:

Taylor: safe

Brooklyn: not safe

Avery: not safe

## Continued on the next page.

## Average Donation

Sarah and Kelly are devoted to donating to several different causes. Both want to receive the prize for the “number 1 donor of the department” they work in. The rules for the prize specify that the winner will be the donor with the maximum average donation, as long as some conditions are satisfied:

1. the winner must have donated to at least 5 different causes;
2. the total amount donated is at least 300 dollars.

If the average is the same, the winner is the one with most donations. Sarah and Kelly want to compare their donations to determine who among them would win the prize (if any). They each take note of their total donation to each cause (as a non-negative integer value) and signify the end of the list with a negative value.

* For the donations below, Sarah wins

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sarah | 100 | 200 | 50 | 100 | 100 | 51 | -10 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kelly | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | -40 |

* For the donations below, Kelly Wins

|  |  |  |  |
| --- | --- | --- | --- |
| Sarah | 500 | 700 | -100 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Kelly | 50 | 100 | 50 | 50 | 50 | -100 |

* For the donations below, no one qualifies for the prize

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sarah | 10 | 200 | 50 | 10 | 10 | 5 | -10 |

|  |  |  |
| --- | --- | --- |
| Kelly | 1000 | -40 |

* For the donations below, they tie

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sarah | 100 | 200 | 50 | 100 | 100 | 50 | -10 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Kelly | 10 | 130 | 50 | 150 | 70 | 190 | -40 |

Your task in this problem is to write a function

void bestDonor(int kellyDonations[], int numKellyDonations,  
 int sarahDonations[], int numSarahDonations)

that takes as input their donations and output the outcome to standard out: either Sarah wins, Kelly wins, they tie, or they do not qualify for the prize.

(A function declared as returning ‘void’ does not return any specific value)

Notice that the arrays have a sentinel value (a negative integer at the end), so you can solve this problem without the size argument:

void bestDonor(int kellyDonations[], int sarahDonations[])