Replit Exerises in Exam Format

Credit to Douglas Harder for all problems

Section 1 - Programming Fundamentals

[1 PTS] 1-1a Finding a bug

Why does this source code not compile?

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int min() {
  std::cout << "Hello World!"
            << std::endl;

  return 0;
}
```

[1 PTS] 1-1b Finding a bug

Why does this source code not compile?

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
  std::cout << "Hello World!"
            << std::endl

  return 0;
}
```

[1 PTS] 1-1c Finding a bug

Why does this source code not compile?

```
#inclde <iostream>

// Function declarations
int main();

// Function definitions
int main() {
  std::cout << "Hello World!"
            << std::endl;

  return 0;
}
```

[1 PTS] 1-1d Finding a bug

Why does this source code not compile?

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
  std::cout "Hello World!" << std::endl;

  return 0;
}
```

[1 PTS] 1-1e Finding a bug

Why does this source code not compile?

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
  std::cout << Hello World! << std::endl;

  return 0;
}
```

[1 PTS] 1-1f Finding a bug

Why does this source code not compile?

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
  std::cout << "Hello World!"" << std::endl;

  return 0;
```

[4 PTS] 1-3 Finding a bug

Each statement in this example has a bug in it. Find the bugs and fix them.

```
int main() {
  // 1
  std::cout << "Planck's constant: "
            << (6.62607004*10^(-34)) << std::endl;
  // 2
  std:cout << 42 << std::endl;
  // 3
  / Print an approximation of 'pi'
  std::cout << 3.142592654 << std::endl;
  // 4
  std::cout << 'Hello world!' << std::endl;
  // 5
  std::cout << '' << std::endl;
  // 6
  std::cout << The answer to the ultimate question is
            << 43 << std::endl;
  // 7
  std::cout << "Please say "Please"." << std::endl;
  // 8
  std::cout << "The Boolean values are "
            << true false << std::endl;
  // 9
  std::cout << "It was the best of times, " <<
            << "it was the worst of times, ..." << std::endl;
  // 10
  std::cout << "Go to the directory C:\Users\dwharder"
            << std::endl;
  // 11
  std::cout << '"' << ''' << std::endl;

  return 0;
}
```

[3 PTS] 1-4 Finding a bug

Some of these identifier names are valid, others are invalid, and some are reserved or keywords in C++. For each identifier that is invalid, reserved, or a keyword, suggest a change that makes it a valid identifier for a local variable.

```
int n;
int double;
int pi2;
int 2pi;
int n0;
int return;
int return_value;
int an_identifier;
int another__identifier;
int _yet_another_identifer_;
int _;
int AnInvalidIdentifier;
int static_assert;
int 0_10_interval;
```

[3 PTS] 1-7 Rounding up

The following function performs integer division by 7

```
int int_div_7( int n ) {
  int m{ 7 };
  return n/m;
}

// These should output 2
std::cout << int_div_7( 14 ) << std::endl;
std::cout << int_div_7( 15 ) << std::endl;
std::cout << int_div_7( 20 ) << std::endl;
// These should output 3
std::cout << int_div_7( 21 ) << std::endl;
std::cout << int_div_7( 22 ) << std::endl;
std::cout << int_div_7( 27 ) << std::endl;
// These should output 4
std::cout << int_div_7( 28 ) << std::endl;
std::cout << int_div_7( 29 ) << std::endl;
```

Write a modified function that will have the following output:

```
// These should output 2
std::cout << div_7( 14 ) << std::endl;
// These should output 3
std::cout << div_7( 15 )  << std::endl;
std::cout << div_7( 20 )  << std::endl;
std::cout << div_7( 21 )  << std::endl;
// These should output 4
std::cout << div_7( 22 )  << std::endl;
std::cout << div_7( 27 )  << std::endl;
std::cout << div_7( 28 )  << std::endl;
// This should output 5
std::cout << div_7( 29 )  << std::endl

int div_7 ( int n ) {
  // Your code here

}
```

[2 PTS] 1-7 Rational division

Write a function which, given a numerator and a denominator, prints a number like 52134/321 as `52134/321 = 162 + 132/321`. You may assume the user is entering valid numerators (0, 1, 2, 3, 4, ...) and valid denominators (1, 2, 3, 4, ...)

For example:

```
print_frac( 52134, 321 );
print_frac( 91, 13 );
```

Would output

```
52134/321 = 162 + 132/321
91/13 = 7 + 0/13
```

```
void print_frac( int num, int den ) {
  // Your code here

}
```

[1 PTS] 1-7a Finding a bug

This program queries the user for a temperture and then asks if the temperature is in either Celsius or Fahrenheit.

If the temperature is in Celsius, then the program prints out the temperature in fahrenheit; otherwise, the temerature is in Fahrenheit, so the program prints out the temperature in celsius.

Find any error(s) in the program and suggest a fix for each.

This question was designed with help from Wani Gupta.

```
int main() {
 while ( true ) {
   double temp{};
   std::cout << "Enter a temperature (Celsius or Fahrenheit): ";
   std::cin >> temp;

   bool in_celsius{};
   std::cout << "Is this temperature in Celsius?" << std::endl;
   std::cout << "  Enter 1 for yes, 0 for no: ";
   std::cin >> in_celsius;

   if ( in_celsius ) {
     //       9
     // T  = --- T  + 32
     //  F    5   C
     temp = (9/5)*temp  + 32;

     std::cout << "The temperature in Fahrenheit is "
               << temp << std::endl;
   } else {
     //       5
     // T  = --- (T  - 32)
     //  C    9    F
     temp = (5/9)*(temp - 32);

     std::cout << "The temperature in Celsius is "
               << temp << std::endl << std::endl;
   }
 }

 return 0;
}
```

[3 PTS] 1-7b Finding a bug

Find and fix the errors in the following code, and find any improvements to make the code more readable.

```
int main() {
  double a{};
  std::cout << "Enter the lower-"
            << "bound: ";
  std::cin  >> a;

  double b{};
  std::cout << "Enter the bound-"
            << "bound: ";
  std::cin  >> b;

  double x1{ a +   b - a/4.0 };
  double x2{ a + 2*b - a/4.0 };
  double x3{ a + 3*b - a/4.0 };

  std::cout << "Five equally-spaced"
            << " points are:"
           << std::endl;

  std::cout << "   " << a
            << ", " << x1
            << ", " << x2
            << ", " << x3
            << ", " << b
            << std::endl;

  return 0;
}
```

[2 PTS] 1-8 Is it odd or even

Write a function that indicates if a number passed by a user is either odd or even.

Be careful about negative numbers.

```
int main() {
int n{};
std::cout << "Enter an integer: ";
std::cin  >> n;

is_even( n );

return 0;
}

void is_even( int n ){
  // Your code here

}
```

[4 PTS] 1-8 Calculating commision

The commission of a sales representative for a particular compnay is as follows:

On the first $20000 sold, the salse representative receives no commission.

Anything over this, the sales rep. receives 8%.

If the sales rep. sells more than $1,000,000, the rep. gets a bonus of $50,000.

Except if the sales rep. sells more than $1,500,000, the rep. instead gets a bonus of $100,000.

The program should request the amount the sales representive sold, and then it should print how much the sales rep is paid, including bonuses.

If the user enters an amount less than zero, just return 0.

```
int main() {
  // Your code here

}
```

[5 PTS] 1-8 Price based on quantity sold

The price for a microcontroller is $12.75 if the customer purchases fewer than 10.

If the customer purchases between 10 and 99, the price is $10.15.

If the customer purchases 100 or more, the price is $9.75

Write a program to query how many microcontrollers are being sold and calculate a cost. You will do this by calculating the cost in cents, not dollars, but you will then print out the result in dollars and cents.

If the amount sold is negative, just return 0

```
int main() {
  // Your code here

}
```

[5 PTS] 1-8 Simple arithmetic test

Write a program that will ask the user to enter two integers and then ask the user what the sum, difference and product of the two numbers is, and checks their answers.

```
int main() {
  // Your code here

}
```

[4 PTS] 1-8 Closest kilometer

Query the user for an integer number of meters and then round that to the closest kilometer. The number of meters should be non-negative (positive or 0), otherwise, return 0.

You will note that integer division always rounds down. You should round to the closest kilometer with the following rule:

If the distance to the closest kilometer is exactly 500 m, then if the number of kilometers is even, then use that distance; otherwise, the number of kilometers is odd, so round up.

Print out the number of kilometers after rounding.

You may have heard the expression: “Even, leave it; odd, raise it.”

```
int main() {
  // Your code here

}
```

[3 PTS] 1-8 Positive, zero, or negative

Write a function that indicates if a number passed by a user is either positive, zero or negative.

```
int main() {
  int n{};
  std::cout << "Enter an integer: ";
  std::cin  >> n;

  // Your code here

}
```

[3 PTS] 1-8 Fast norm calculation

```
Normally, calculating the length or 2-norm
                             / x \
of a two-dimensional vector |     |
                             \ y /
(also called the Euclidean length of the vector) requires one to calculate

              _________
             /  2    2
           \/  x  + y  ,
 which requires a square root. However, if we define

                        _______________
                  ___  /           ___
   a = -6 - 4 \/ 2  + 2 \/ 20 + 14 \/ 2
     = 0.96043387010341996525


          ___   ___
   b = \/ 2  (\/ 2  - 2)

                          _____________
         /         ___    /         ___  \
      * (  3+2 \/ 2  - \/20 + 14 \/ 2    )
         \                              /
     = 0.39782473475931601382

then if x >= y, the 2-norm can be approximated by
                 a*x + b*y
where the maximum relative error is given by:
              1 - a = 0.03956613,
so less than 3.96%.

Note, this approximation is has approximately 1% less relative error than using 0.96*x + 0.4*y, which has a
maximum relative error of 4%.

Implement a function that calculates this approximation for any real two-dimensional vector.
double fast_norm( double x, double y ) {
  // Your code Here

}
```

[4 PTS] 1-8 Calculating income tax

According to the Canadian Revenue Agency, the tax brackets for those paying Federal income tax brackets are as follows:

$49,020 or less - 15%

$49,020 to $98,040 - 20.5%

$98,040 to $151,978 - 26%

$151,978 to $216,511 - 29%

More than $216,511 - 33%

What this says is that every taxpayer pays only 15% on the first $49 020, even if you make $1 million in that year.

If you make $50 000, you pay 15% on the first $49 020, and 20.5% on the remaining $980.

If you make $100 000, you pay 15% on the first $49 020, 20.5% on the next $49 020, and 26% on the remaining $1960.

If you make $200 000, you pay 15% on the first $49 020, 20.5% on the next $49 020, 26% on the next $53 938, and 29% on the remaining $48 022.

To calculate 15%, use the calculation (amount*15)/100

To calculate 20.5%, use the calculation (amount*205)/1000

Write a program that calculates the income tax for a given income

```
int main() {
  int income{};
  std::cout << "Enter your income: ";
  std::cin  >> income;

  // Your code here

  std::cout << "You owe " << income
            << " in taxes." << std::endl;

  return 0;
}
```

[3 PTS] 1-8 Second largest of four

Given four integers, a, b, c and d, determine which is the second-largestvalue of the four. We will define the ‘second-largest’ as the number that appears next to the largest when the list is sorted, so if all four values are equal, both the largest and the second-largest are that value.

Examples:

Of 1 1 2 2, the second-largest is 2

Of 1 1 2 3, the second-largest is 2

Write a function that queries the user for four integers and then prints the second-largest according to the definition above.

```
int second_largest() {
  // Your code here

}
```

[4 PTS] 1-8 Printing a complex number

An integer complex number is a number of the form 'm + nj' where 'm' and 'n' are integers.

If you print 'm + nj' as

```
std::cout << m << " + " << n << "j" << std::endl;
```

You will get some awkward outputs, such as

0 + −3j

0 + 0j

−3 + −1j

We want a nicer printing of a complex number that follows these rules:

```
 If 'n = 0', print m
 If 'm = 0', print nj,
             unless n =  1, just print j,
                 or n = -1, just print -j
 If 'n = -1', print m - j
 If 'n <= -2', print m - (-n)j,
              so for example 3 - 5j
 Otherwise, print m + nj
             unless 'n = 1', just print 'm + j'
```

Write a function that prints complex numbers as described

```
void complex_print( int m, int n ) {
  // Your code here

}
```

[4 PTS] 1-8 A simple calculator

Write a program that queries the user for two integer values and then asks the user to enter an integer between 1 and 4 indicating that the operation should be addition, subtraction, multiplication or division.

If the user enters any other integer, indicate that it is an invalid opertion.

If the integers are 7 and 12, then the output should be

7 + 12 = 19

7 - 12 = −5

7 x 12 = 84

7 / 12 = 0 with a remainder of 7

If the remainder is zero, just print out the result of the division, e.g.,

20/5 = 4

```
int main() {
  // Your code here

}
```

[4 PTS] 1-8 A rational calculator

Write a program that queries the user for two raltional numbers and then asks the user to enter an integer between 1 and 4 indicating that the operation should be addition, subtraction, multiplication or division.

If the user enters any other integer, indicate that it is an invalid opertion.

If the rational numbers are a/b and c/d, then the output should be

```
a/b + c/d = (a*d + b*c)/(b*d)
a/b - c/d = (a*d - b*c)/(b*d)
a/b * c/d = (a*c)/(b*d)
a/b / c/d = (a*d)/(b*c)
```

You will print out the numerator and the denominator of the result with a "/" between them.

Don't worry about lowest terms, so for example, 1/2 + 1/2 will come out as 4/4; however, if the denominator is negative, negate both the numerator and denominator

```
int main() {
  // Your code here

}
```

[3 PTS] 1-8 Shortest distance

Suppose we have two locations A and D and there are two other locations B and C, and you can get from any location to any other. There is a distance between any two points, and that distances is positive.

Write a function that takes the 6 distances, and then determine the shortest distance between A and D, allowing for the possibility that one can stop at B or C or both, first.

If any of the distances are not positive, return 0.

```
double shortest_distance( double AB, double AC, double AD, double BC, double BD, double CD ) {
  // Your code here

}
```

[1 PTS] 1-8 Finding a bug

In calculating the arctangent of value, you must be aware of which quadrant the point is in, for the arctangent function does not return all possible values of an angle, for the following is true:

The angle of the point (4, 4) is 45 degrees, and to calculate this, we must calculate atan(4/4)

The angle of the point (-4, −4) is 225 degrees, but if we calculate atan(−4/-4), this gives us the same value as atan(4/4), which is 45 degress.

Thus, we must be careful of which quadrant we are in before calculating the result. Also, this program is to print the closest integer degree, while std::atan returns radians. The integer degree must be a value between 0 and 359 degrees: - the angle may never be negative or greater than 359

Find and suggest fixes for all the bugs in the following program. Be sure to fix esthetic errors, too.

```
int main() {
  while ( true ) {
    double x{};
    std::cout << "Enter an x value: ";
    std::cin >> x;

    double y{};
    std::cout << "ENter a y value: ";
    std::cin >> y;

    double result{};
    int    degrees{};

    if ( y >= 0.0 ) {
      // We are in the first or second quadrants,    rise
      //        so use the fact that tan( theta ) = ------
      //                                            run
      result = std::atan( y/x );
    } else {
      result = M_PI + std::atan( y/x );
    }

    // The std::atan function returns a value between [0, M_PI),
    // so the above calculation returns a value in [0, 2 pi)
    //  - There are 360 degrees in a circle, so we calculate:
    degrees = 360/M_PI*result;

    std::cout << "The angle of (" << x << ", " << y << " ) is "
              << degrees << " degrees" << std::endl << std::endl;
  }

  return 0;
}
```

[3 PTS] 1-9 Counting years

Most calendar systems have a chosen year marked as '1', such as 1 CE or 1 AH, the next year is 2 CE or 2 AH, but the previous year is 1 BCE or 1 BH, respectively.

We will represent years that are CE or AH with positive integers, and years that are BCE or BH as negative integers. 0 is not a valid year.

Write a function that takes two years, where the second year must be greater than the first. If not, advise the user and 'return 0;'

Calculate the number of years between the two, including the two given years. Thus, for example, the number of years between 1970 and 1980 is 11.

If either year is '0', advise the user that '0' is not a valid year and 'return 0;'

This calculation, however, will change differ if the first year is negative and the second is positive. Determine the correct formula and implement it.

```
int year_count( int year1, int year2 ) {
  // Your code here

}
```

[3 PTS] 1-9 Passing conditions

A course is made up of four modules, each scored out of 25. The final grade is the sum of these four modules.

To pass the course you must:
1. Pass at least three modules, and
2. Receive a minimum grade of 50.

Write a function that takes five grades and then returns if the course is passed or failed.

If any entered grade is less than 0 or greater than 25, just return false.

```
bool is_course_passed( int grade1, int grade2, int grade3, int grade4 ) {
  // Your code here

}
```

[2 PTS] 1-9 Leap years

A year is a leap year if it is divisible by four, but not divisible by 100, unless it is also divisible by 400.

Write a function that takes an integer year:
- If the integer is 0 or negative, return false.
- If the integer is positive, return true if it is a leap year.

```
bool is_leap_year( int year ) {
  // Your code here

}
```

[2 PTS] 1-9 Past specified date

A date is stored in three variables, 'year', 'month' and 'day'.

Write a program to query the user for a year, month and day, and indicate whether the day entered is before, equal to or after the stored date.

For the purposes of this program, you may assume all months have 31 days.

If the user enters a year that is 0, return 0.

If the user enters a month other than 1 through 12, return 0.

If the user enters a day that is 0, negative, or greater than 31, return 0.

Otherwise print "future" if it is before the stored date, "same day" if it is the same date, and "past" if the date is past the stored date.

```
int main() {
  int year{ 2022 };
  int month{ 9 };
  int day{ 13 };

  int user_year{};
  int user_month{};
  int user_day{};

  // Your code here

}
```

[2 PTS] 1-9a Finding a bug

1. What is the issue with the coding style of this code.
2. Do the conditions actually test for what is being described?

```
int main() {
  int m{};
  int n{};

  std::cout << "Enter a first integer: ";
  std::cin  >> m;
  std::cout << "Enter another integer: ";
  std::cin  >> n;

  if ( m > 0 && n < -1 || n > 1 ) {
    std::cout << "Your first integer is positive "
              << "and your second integer is not "
              << "one of -1, 0 or 1"
              << std::endl;
  }

  if ( m < 0 || n > 0 && n > m ) {
    std::cout << "Your second integer is greater "
              << "than the first, and either the "
              << "first is negative or the second "
              << "is positive." << std::endl;
  }

  if ( m < 0 || n < 0 && m > 0 || n > 0 ) {
    std::cout << "One of your integers is negative "
              << "and the other is positive." << std::endl;
  }

  return 0;
}
```

[2 PTS] 1-9b Finding a bug

What is the bug or are the bugs in the following code, and how would you fix it or them?

```
int main() {
  int m{};
  int n{};

  std::cout << "Enter an integer 'm': ";
  std::cin  >> m;

  std::cout << "Enter an integer 'n': ";
  std::cin  >> n;

  if ( ((m < 0) && (n > 0))
    || ((n > 0) && (m < 0)) ) {
    std::cout << m << " and " << n
              << " have opposite signs."
              << std::endl;
  } else if ( (m == 0) || (n == 0) ) {
    std::cout << "One of 'm' and 'n' is zero."
              << std::endl;
  } else if ( (m == 0) && (n == 0) ) {
    std::cout << "Both 'm' and 'n' are zero."
              << std::endl;
  } else {
    std::cout << m << " and " << n
              << " have the same sign."
              << std::endl;
  }

  return 0;
}
```

[2 PTS] 1-9c Finding a bug

```
 Can you get the output to be:
   The values ? and ? are equal!
   Your two values are equal.

Next, can you get the output to be:
   The values ? and ? are equal!
   The third doesn't equal the first or equals the second.

If so, what are values of x, y, and z;
and if not, why not?
```

```
bool verbose_equality( double a, double b );

int main() {
  double x{};
  std::cout << "Enter a real number: ";
  std::cin  >> x;

  double y{};
  std::cout << "Enter another real number: ";
  std::cin  >> y;

  if ( verbose_equality( x, y ) ) {
    std::cout << "Your two values are equal."
              << std::endl;
    return 0;
  }

  double z{};
  std::cout << "Enter a third real number: ";
  std::cin  >> z;

  if ( (x != z) || verbose_equality( z, y ) ) {
    std::cout << "The third doesn't equal the first or equals the second."
              << std::endl;
  }

  return 0;
}

bool verbose_equality( double a, double b ) {
  if ( a == b ) {
    std::cout << "The values " << a
              << " and " << b
              << " are equal!" << std::endl;
    return true;
  } else {
    return false;
  }
}
```

[3 PTS] 1-10 Rational division

Write a function that takes an integer numerator and denominator and prints a rational number in the style of the following

```
52134/321 = 162 + 132/321
```

```
91/13 = 7
```

Your program should not print the fractional part if the fractional part is zero.

You may assume the user is entering valid numerators (0, 1, 2, 3, 4, ...) and valid denominators (1, 2, 3, 4, ...)

```
void print_rational( int num, int den ) {
  // Your code here

}
```

[2 PTS] 1-10a Finding a bug

This program asks the user for three integers and then swaps the entries around so as to ensure that 'c' is assigned the largest possible value.

This code has poor coding practice, weaknesses, and possibly a bug. Your job is to identify the poor coding practcises, the weaknesses, and then to determine if there is in fact a bug.

```
int main() {
  int a;
  int b;
  int c;

  std::cout << "Enter a number: ";
  std::cin  >> a;
  std::cout << "Enter a second number: ";
  std::cin  >> b;
  std::cout << "Enter a final number: ";
  std::cin  >> c;

  if ( a >= b && a >= c ) {
    //Swap 'a' and 'c'
    int b{ a };
    a = c;
    c = b;
  } else if ( b >= c ) {
    //Swap 'b' and 'c'
    int a{ b };
    c = a;
    b = c;
  }

  std::cout << "The largest is " << c
            << std::endl;

  return 0;
}
```

[1 PTS] 1-11a Writing a for loop

Fill in the program so the first for loop prints the integers from 0 to 9, and then the second for loop prints the integers from 2 to 7

```
int main() {
  for (    ;    ;    ) {
    std::cout <<  << " ";
  }

  std::cout << std::endl;

  for (    ;    ;    ) {
    std::cout <<  << " ";
  }

  std::cout << std::endl;

  return 0;
}
```

[2 PTS] 1-11b Writing a for loop

Write function that takes two integers and prints all the integers starting with the first up to and including the second.

If the second is less than the first, then print nothing

```
void print_up_to( int m, int n ) {
  // Your code here

}
```

[2 PTS] 1-11c Writing a for loop

Write function that takes two integers and prints all the integers starting with the lesser of the two up to and including the greater of the two.

```
void print_up_to( int m, int n ) {
  // Your code here

}
```

[2 PTS] 1-11d Writing a for loop

Write a function that takes two integers and prints the numbers from the first to second, going either up or down.

```
void print_from_m_to_n( int m, int n ) {
  // Your code here

}
```

[3 PTS] 1-11e Writing a for loop

Write a function that takes two integers and

1. Prints the from the minimum of the two, and goes to the maximum but with a step size of 2, so to go from 2 to 7, it would print `2 4 6`
2. Again prints from the minimum to the maximum, but only prints the even numbers.

```
void print_by_twos( int minimum, int maximum ) {
  // Your code here

}
```

[1 PTS] 1-11 Calculting integer powers

Write a function that takes a real number x and an integer exponent n and returns the result

```
double calculate_exponent( double x, int n ) {
  // Your code here

}
```

[1 PTS] 1-11 Calculating a factorial

Write a function which calculates the factorial of the given number

```
unsigned int factorial( unsigned int n ) {
  // Your code here

}
```

[1 PTS] 1-11 Calculating a double factorial

Implement a program that takes an integer 'n' and then returns 'n!!'; that is, the double factorial. This is used in the occasional engineering model.

If n < 0, just return 0 and do nothing.

If 'n' is even, then `n!! = n(n - 2)(n - 4)...4 x 2`

If 'n' is odd, then `n!! = n(n - 2)(n - 4)...5 x 3 x 1`

```
int double_factorial( int n ) {
  // Your code here

}
```

[1 PTS] 1-11 Summing a range of numbers

Write a function that takes two integers and calculates the sum of all integers between the two, inclusive.

```
int range_sum( int m, int n ) {
  // Your code here

}
```

[2 PTS] 1-11 Finding the gcd

Write a function that takes two non-negative integers and returns the greatest common divisor with the following requirements

`gcd( 0, 0 ) = 0`

`gcd( 0, n ) = n`

`gcd( m, 0 ) = m`

Otherwise, the gcd is the largest positive integer that divides both.

```
unsigned int gcd( unsigned int m, unsigned int n ) {
  // Your code here

}
```

[2 PTS] 1-27a Counting one bits

Write a function that counts the number of '1' bits in the binary representation of a long.

```
unsigned int count_ones( unsigned long n ) {
  // Your code here

}
```

[5 PTS] 1-27d Longest run of ones

Write three functions that return:
1. The longest sequence of consecutive ones; that is, the longest run of ones.
2. The longest run of zeros.
3. The longest run of the same bit.

```
unsigned int longest_run_of_ones( unsigned long n );
unsigned int longest_run_of_zeros( unsigned long n );
unsigned int longest_run( unsigned long n );
```

[2 PTS] 1-27g Is power of two

Write a function that takes an integer and returns true if it is a power of two, or false otherwise

```
bool is_power_of_two( unsigned long n ) {
  // Your code here

}
```

[2 PTS] 1-33 Is in array

Write a function that takes an array and a value n and returns true if n is in the array, or false otherwise

```
bool is_in_array( int n, int array[], std::size_t capacity ) {
  // Your code here

}
```

[4 PTS] 1-33 Is an array within an array

Write a function that determines if all the entries in array1 appear in the same order within array2. If so, return the index of array2 where the first instance of array1 starts, otherwise return capacity2

```
std::size_t first_in_array( int array1[], std::size_t capacity1, int array2[], std::size_t capacity2 ) {
  // Your code here

}
```