Replit Exerises in Exam Format - Answers

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;
}
```

The function that is executed when the program is compiled must be called 'main', so the definition of the function

`int min() {`

defines a function called "min", not "main"

[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;
}
```

Every statement requires a semi-colon at the end of it and the statement

`std::cout << "Hello world!" << std::endl`

is missing such a semi-colon.

As far as the compiler is concerned, the statement continues with the next line,

`std::cout << "Hello world!" << std::endl return 0;`

and this makes no sense.

[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;
}
```

The pre-processor directive is

`#include`

This question has that word mispelled.

[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;
}
```

When printing to the screen, you need to include << operators between the `std::cout` and anything being printed.

[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;
}
```

The string of characters “Hello world!” is not something that the compiler understands as English. Strings of characters (or ‘strings’) must enclosed by an opening and a closing double quote.

The compiler is trying to treat the characters ‘Hello’ and ‘World’ as parts of the program, and the trailing ! is interpreted as an operator, but not the factorial function: world! is an incorrect use of ‘!’ and not ‘world’ factorial. There is no factorial operator in C++.

[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;
```

The ‘block of statements’ that comprise the ‘main’ function begin with the opening brace ‘{‘ and go up until the matching closing brace ‘}’. When the compiler did not find a closing brace, it issued an error.

[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;
}
```

1. The 6.62607004*10^(-34) should be 6.62607004e-34
2. We should have std::cout, not std:cout
3. There is only one / for the comment. Note that you can use quotes and double-quotes in a comment and not worry about their use. You may have missed the other bug: 3.142592654 should be 3.141592654
4. A single quote is being used for a string: you should use double quotes.
5. Single quotes are used to identify a character literal. There is no such thing as an empty character. There is, however, such a thing as an empty string (a string with no characters), which is represented by “”.
6. The string needs double-quotes around it, and the answer to the ultimate question is actually 42.
7. You must escape the double quote in a string: `"Please say \"Please\"."`
8. You need a ‘<<’ between the ‘true’ and the ‘false’, but better yet, you should add a space or intermediate text:

   ```
   << true << " and " << false << std::endl;
   ```

9. There is a ‘<<’ both at the end of the line and at the start of the next line. There should only be one ‘<<’ operator, either at the end of the first line, or this author’s preference, at the start of the next line.
10. You must escape the backslash character in a string: `"Go to the directory C:\\Users\\dwharder"`
11. For a literal character, you must escape the single quote, but it is optoinal to escape a double quote, so both of these are valid:

    ```
    std::cout << '\"' << '\'' << std::endl;
    ```

    ```
    std::cout << '"' << '\'' << std::endl;
    ```

[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;
```

```
int n;              A valid identifier

int double;         Invalid: 'double' is a keyword.
                    Consdier using long_intetger

int pi2;            A valid identifier

int 2pi;            Invalid: an identifier cannot
                    start with a digit. Consider
                    using 'two_pi' or 'pi_2'
int n0;             A valid identifier

int return;         Invalid: 'return' is a keyword.
                    Consider using 'return_value'
int return_value;   A valid identifier.

int an_identifier;  A valid identifier.
int another__identifier;
                    A valid-but-reserved identifer.
                    Consider using 'another_identifier'
                    All identifiers with adjacent
                    underscores are reserved.
int _yet_another_identifer_;
                    A valid-but-reserved identifier.
                    Consider using
                         'yet_another_identifer_'
                    All identifiers starting with an
                    underscore are reserved.

int _;              A valid identifer, but potentially
                    reserved; however, reserved
                    identifiers may 'begin' with an
                    underscore... Anyway, don't use
                    an underscore as an identifier...
int AnInvalidIdentifier;
                    Despite the name,
                    a valid identifier.
int static_assert;
                    Invalid: 'static_assert' is a
                    keyword in C++.
int 0_10_interval;
                    Invalid: an identifier cannot
                    start with a digit. Use, for
                    example, interval_0_10 instead.
```

[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

}
```

```
int div_7( int n ) {
  // Initialise variables
  int m{ 7 };
  int result;

  // Find the integer division of n by 7
  result = n/m;

  // Add 1 to the result if n isn't a multiple of 7
  if ( n%m != 0 ) {
    result += 1;
  }

  return result;
}
```

[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

}
```

```
void print_frac( int num, int den ) {
  // Calculate the result of integer division
  int int_div{ num/den };

  // Output resembles example output
  std::cout  << num << '/' << den << " = "
             << int_div << " + "
             << num - int_div*den << '/' << den
             << std::endl;
  return;
}
```

[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;
}
```

The program treats `9/5` and `5/9` as integer division, rounding the result to 1 and 0 respectively.

Since both parameters of the division are integers, c++ assumes that the output should also be an integer, and returns an integer by rounding down the result of the division.

A fix would be to replace `9/5` and `5/9` with `9/5.0` and `5/9.0` to indicate that the result should also be a floating point value.

[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;
}
```

The calculations for `x1, x2, x3` are incorrect and should be

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

Combining the `cout` statements split across multiple lines would improve readability

```
  std::cout << "Enter the lower-bound: ";
```

The second print statement says “bound-bound” instead of “higher-bound”

[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

}
```

```
void is_even( int n ){
  // Find out if n is even
  bool is_n_even{ n%2 == 0 };

  // Output to the user whether n is even or odd
  if ( is_n_even ) {
    std::cout << n << " is even." << std::endl;
  } else {
    std::cout << n << " is odd." << std::endl;
  }

  return;
}
```

[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

}
```

```
int main() {
  // Receive and store an input for the amount sold
  int sales{};
  std::cout << "What was amount sold by the representative: ";
  std::cin  >> sales;

  // Handle the case of a negative input values
  if ( sales < 0 ) {
    return 0;
  }

  int commission{ 0 };

  // Find the commission from the amount sold
  if ( sales > 20000 ) {
    commission += ((sales - 20000)*8) / 100;
  }

  // Add the bonus to the commission
  if ( sales >= 1000000 ) {
    commission += 50000;
  }

  if ( sales >= 1500000 ) {
    commission += 50000;
  }

  // Output the commission to the user
  std::cout << "The rep's commission with bonuses is "
            << commission << std::endl;

  return 0;
}
```

[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

}
```

```
int main() {
  // Receive and store an input for number of microcontrollers sold
  int amount_sold{};
  std::cout << "Enter the number of microcontrollers sold: ";
  std::cin  >> amount_sold;

  // Handle the case of a negative input
  if ( amount_sold < 0 ) {
    return 0;
  }

  int cost{};

  // Find the total cost based on the amount sold
  if ( amount_sold < 10 ) {
    cost = 1275*amount_sold;
  } else if ( amount_sold < 100 ) {
    cost = 1015*amount_sold;
  } else {
    cost =  975*amount_sold;
  }

  // Output the cost to the user
  std::cout << "The cost is " << (cost/100) << ".";

  // Handle cos with fewer than 10 cents
  cost %= 100;

  if ( cost == 0 ) {
    std::cout << "00" << std::endl;
  } else if ( cost < 10 ) {
    std::cout << "0" << cost << std::endl;
  } else {
    std::cout << cost << std::endl;
  }

  return 0;
}
```

[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

}
```

```
int main() {
  // Receive and store an input of two integers
  int m, n;
  std::cout << "Enter two integers: ";
  std::cin  >> m >> n;

  int users_answer{};
  int correct_answer{};

  // Output the question to the user
  std::cout << "What is " << m " + " << n << "? ";
  // Receive and store the user's answer
  std::cin  >> users_answer;
  // Calculate the correct answer
  correct_answer = m + n;

  // Check the user's answer against the correct answer
  // Output whether the user is correct or incorrect
  if ( users_answer == correct_answer ) {
    std::cout << "You are correct." << std::endl;
  } else {
    std::cout << "You are incorrect: the answer is "
              << correct_answer << std::endl;
  }

  std::cout << "What is " << m " - " << n << "? ";
  std::cin  >> users_answer;

  correct_answer = m - n;

  if ( users_answer == correct_answer ) {
    std::cout << "You are correct." << std::endl;
  } else {
    std::cout << "You are incorrect: the answer is "
              << correct_answer << std::endl;
  }

  std::cout << "What is " << m " x " << n << "? ";
  std::cin  >> users_answer;

  correct_answer = m*n;

  if ( users_answer == correct_answer ) {
    std::cout << "You are correct." << std::endl;
  } else {
    std::cout << "You are incorrect: the answer is "
              << correct_answer << std::endl;
  }

  return 0;
}
```

[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

}
```

```
int main() {
  int distance_m{};
  std::cout << "Enter a distance in meters: ";
  std::cin  >> distance_m;

  if ( distance_m < 0 ) {
    return 0;
  }

  int distance_km{ distance_m/1000 };

  // Determine the remainder
  int remainder_m{ distance_m%1000 };

  if ( remainder_m > 500 ) {
    // If the remainder is is greater
    // than 500 m, then round up the
    // the next highest number of
    // kilometers.
    ++distance_km;
  } else if ( remainder_m == 500 ) {
    // If the remainder is exactly 500 m,
    // then round up if the distance in
    // km is odd, and round down if it
    // is even.
    if ( distance_km%2 == 1 ) {
      ++distance_km;
    }
  }

  std::cout << "The distance is "
            << distance_km << " km"
            << std::endl;

  return 0;
}
```

[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

}
```

Because there are three possibilities, you will need a cascading conditional statement with two consequent blocks and one alternative.

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

  if ( n > 0 ) {
    std::cout << n << " is positive." << std::endl;
  } else if ( n == 0 ) {
    std::cout << n << " is zero." << std::endl;
  } else {
    std::cout << n << " is negative." << std::endl;
  }

  return 0;
}
```

DO NOT include a third condition and no alternative block:

```
if ( n > 0 ) {
  std::cout << n << " is positive." << std::endl;
} else if ( n == 0 ) {
  std::cout << n << " is zero." << std::endl;
} else if ( n < 0 ) {
  std::cout << n << " is negative." << std::endl;
}
```

[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

}
```

Hint: A vector can have both positive and negative values, and there is no guarantee the first entry is the largest in absolute value.

```
double fast_norm( double x, double y ) {
  double a{ 0.96043387010341996525 };
  double b{ 0.39782473475931601382 };

  // If either 'x' or 'y' is negative, make them positive
  if ( x < 0 ) {
    x = -x;
  }
  if ( y < 0 ) {
    y = -y;
  }

  // The formula requires that 'a' is multiplied by the larger of the two.
  if ( x >= y ) {
    return a*x + b*y;
  } else {
    return a*y + b*x;
  }
}
```

[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;
}
```

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

  int income_tax{ 0 };
  //        0,  49020     49020
  //    49020,  98040     49020
  //    98040, 151978     53938
  //   151978, 216511     64533
  //     More than $216 511   33%

  // Pay 15% on the first $49020
  if ( income <= 49020 ) {
    income_tax += (income*15)/100;
  } else {
    income_tax += (49020*15)/100;
  }

  income -= 49020;

  if ( income > 0 ) {
    // Pay 20.5% on the next $49020
    if ( income <= 49020 ) {
      income_tax += (income*205)/1000;
    } else {
      income_tax += (49020*205)/1000;
    }

    income -= 49020;
  }

  if ( income > 0 ) {
    // Pay 26% on the next $53938
    if ( income <= 53938 ) {
      income_tax += (income*26)/100;
    } else {
      income_tax += (53938*26)/1000;
    }

    income -= 53938;
  }

  if ( income > 0 ) {
    // Pay 29% on the next $64533
    if ( income <= 64533 ) {
      income_tax += (income*29)/100;
    } else {
      income_tax += (64533*29)/100;
    }

    income -= 64533;
  }

  if ( income > 0 ) {
    // Everything else is taxed at 33%
    income_tax += (income*33)/100;
  }

  std::cout << "You owe " << income_tax
            << " 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

}
```

```
int second_largest() {
  int a{};
  int b{};
  int c{};
  int d{};
  int largest{};
  int second{};

  std::cout << "Enter an integer: ";
  std::cin  >> a;
  std::cout << "Enter a second integer: ";
  std::cin  >> b;

  if ( a >= b ) {
    largest = a;
    second = b;
  } else {
    largest = b;
    second = a;
  }

  std::cout << "Enter a third integer: ";
  std::cin  >> c;

  if ( c >= largest ) {
    second = largest;
    largest = c;
  } else if ( c > second ) {
    second = c;
  }

  std::cout << "Enter a fourth integer: ";
  std::cin  >> d;

  if ( d >= largest ) {
    second = largest;
    largest = d;
  } else if ( d > second ) {
    second = d;
  }

  return second;
}
```

[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

}
```

```
void complex_print( int m, int n ) {
  if ( n == 0 ) {
    std::cout << m << std::endl;
  } else if ( m == 0 ) {
    if ( n == 1 ) {
      std::cout << "j" << std::endl;
    } else if ( n == -1 ) {
      std::cout << "-j" << std::endl;
    } else {
      std::cout << n << "j" << std::endl;
    }
  } else {
    // At this point, neither 'm' nor 'n' are zero
    if ( n <= -2 ) {
      std::cout << m << " - " << (-n) << "j" << std::endl;
    } else if ( n == -1 ) {
      std::cout << m << " - j" << std::endl;
    } else if ( n == 1 ) {
      std::cout << m << " + j" << std::endl;
    }  else {
      std::cout << m << " + " << n << "j" << std::endl;
    }
  }

  return;
}
```

[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

}
```

```
int main() {
  int m{};
  std::cout << "Enter the left-hand operand: ";
  std::cin  >> m;
  int n{};

  std::cout << "Enter the right-hand operand: ";
  std::cin  >> n;

  int choice{};
  std::cout << "Enter 1 for addition," << std::endl;
  std::cout << "      2 for subtraction," << std::endl;
  std::cout << "      3 for multiplication, and" << std::endl;
  std::cout << "      4 for division: " << std::endl;
  std::cin  >> choice;

  if ( choice == 1 ) {
    std::cout << m << " + " << n << " = "
              << (m + n) << std::endl;
  } else if ( choice == 2 ) {
    std::cout << m << " - " << n << " = "
              << (m - n) << std::endl;
  } else if ( choice == 3 ) {
    std::cout << m << " x " << n << " = "
              << (m*n) << std::endl;
  } else if ( choice == 4 ) {
    std::cout << m << " / " << n << " = "
              << (m/n);

    if ( m%n == 0 ) {
      std::cout << std::endl;
    } else {
      std::cout << " with a remainder of " << (m%n) << std::endl;
    }
  } else {
    std::cout << "Invalid operation: " << choice << std::endl;
  }

  return 0;
}
```

[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

}
```

```cpp
int main() {
  int num_m{};
  int den_m{};
  std::cout << "Enter the left-hand numerator: ";
  std::cin  >> num_m;
  std::cout << "Enter the left-hand denominator: ";
  std::cin  >> den_m;

  int num_n{};
  int den_n{};
  std::cout << "Enter the right-hand numerator: ";
  std::cin  >> num_n;
  std::cout << "Enter the right-hand denominator: ";
  std::cin  >> den_n;

  int choice{};
  std::cout << "Enter 1 for addition," << std::endl;
  std::cout << "      2 for subtraction," << std::endl;
  std::cout << "      3 for multiplication, and" << std::endl;
  std::cout << "      4 for division: " << std::endl;
  std::cin  >> choice;

  int num{};
  int den{};

  if ( choice == 1 ) {
    num = num_m*den_n + num_n*den_m;
    den = den_m*den_n;
  } else if ( choice == 2 ) {
    num = num_m*den_n - num_n*den_m;
    den = den_m*den_n;
  } else if ( choice == 3 ) {
    num = num_m*num_n;
    den = den_m*den_n;
  } else if ( choice == 4 ) {
    num = num_m*den_n;
    den = den_m*num_n;
  }

  // If the denominator is negative,
  //   negate both the numerator and the denominator
  if ( den < 0 ) {
    num *= -1;
    den *= -1;
  }

  std::cout << num << "/" << den << std::endl;

  return 0;
}
```

[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

}
```

```
double shortest_distance( double AB, double AC, double AD, double BC, double BD, double CD ) {
  // Start by assuming the distance between A and D is the shortest
  double distance{ AD };

  // See if any other paths are shorter
  if ( (AB + BD) < distance ) {
    distance = AB + BD;
  }

  if ( (AC + CD) < distance ) {
    distance = AC + CD;
  }

  if ( (AB + BC + CD) < distance ) {
    distance = AB + BC + CD;
  }

  if ( (AC + BC + BD) < distance ) {
    distance = AC + BC + BD;
  }

  return distance;
}
```

[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;
}
```

Here is a list of all bugs

1. The mispelling of “ENter”
2. The space after the closing parenthsis of the coordinate in the answer.
3. std::atan returns an angle in (-pi/2, pi/2) and not [0, pi) and therefore, the conditional statement must be rewritten for the first and fourth coordinates.
4. If ‘x == 0’, then ‘y/x’ will result in plus or minus infinity, and thus, we shouldeal with these speical cases separately.
5. The code above truncates and does not round. To round, you can either use the std::round function that is implemented in a library, or you can simply add however many radians there are in 1/720 degrees.
6. The fix in Error 5, however, introduces another issue:the angle may end up being 360 degrees, so this special case, too, must be checked for.

[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

}
```

```
int year_count( int year1, int year2) {
  if ( year1 == 0 ) {
    std::cout << "0 is not a valid year" << std::endl;
    return 0;
  }

  if ( year2 == 0 ) {
    std::cout << "0 is not a valid year" << std::endl;
    return 0;
  }

  if ( year1 > year2 ) {
    std::cout << "The first year should not be after the second"
              << std::endl;
    return 0;
  }

  int delta{};

  if ( year1 < 0 && year2 > 0 ) {
    delta = year2 - year1;
  } else {
    delta = year2 - year1 + 1;
  }

  std::cout << "There are " << delta
            << " years between " << year1
            << " and " << year2
            << " inclusive." << std::endl;

  return 0;
}
```

[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

}
```

```
bool is_course_passed( int grade1, int grade2, int grade3, int grade4 ) {
  if ( (grade1 < 0) || (grade1 > 25) ) {
    return false;
  }

  if ( (grade2 < 0) || (grade2 > 25) ) {
    return false;
  }

  if ( (grade3 < 0) || (grade3 > 25) ) {
    return false;
  }

  if ( (grade4 < 0) || (grade4 > 25) ) {
    return false;
  }

  if ( (grade1 + grade2 + grade3 + grade4) < 50 ) {
    return false;
  } else {
    int pass_count{ 0 };

    if ( grade1 >= 13 ) {
      ++pass_count;
    }

    if ( grade2 >= 13 ) {
      ++pass_count;
    }

    if ( grade3 >= 13 ) {
      ++pass_count;
    }

    if ( grade4 >= 13 ) {
      ++pass_count;
    }

    if ( pass_count >= 3 ) {
      return true;
    } else {
      return false;
    }
  }
}
```

[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

}
```

```
bool is_leap_year( int year ) {
  if ( year <= 0 ) {
    return false;
  } else if ( (year%4 == 0) && ((year%100 != 0) || (year%400 == 0)) ) {
    return true;
  } else {
    return false;
  }

  return false;
}
```

[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

}
```

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

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

  std::cout << "Enter a year: ";
  std::cin  >> user_year;

  if ( user_year == 0 ) {
    return 0;
  }

  std::cout << "Enter a month (1-12): ";
  std::cin  >> user_month;

  if ( (month <= 0) || (month >= 13) ) {
    return 0;
  }

  std::cout << "Enter a day (1-31): ";
  std::cin  >> user_day;

  if ( (user_day <= 0) || (user_day >= 32) ) {
    return 0;
  }

  if ( (user_year < year) || (
    (user_year == year) && (user_month < month)
  ) || (
    (user_year == year) && (user_month == month)
                        && (user_day < day)
  ) ) {
    std::cout << "future" << std::endl;
  } else if ( (user_year == year) && (user_month == month)
                                  && (user_day == day) ) {
    std::cout << "same date" << std::endl;
  } else {
    std::cout << "past" << std::endl;
  }

  return 0;
}
```

[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;
}
```

```
Coding style:
  Apart from arithmetic expressions using
  only addition and multiplication, always
  include parentheses around operators.

Thus, (a < b) || ((c <= d) && (e > f))
is appropriate as the operands of the logical OR
are clearly (a < b) and ((c <= d) && (e > f))
and the operands of the logical AND are
clearly (c <= d) and (e > f)

Neither expression evaluates to what is
desired, as the first is equivalent to:
 if ( ((m > 0) && (n < -1)) || (n > 1) ) {
and the second is equivalent to
 if ( (m < 0) || ((n > 0) && (n > m)) ) {
while the third is equivalent to
 if ( (m < 0) || ((n < 0) && (m > 0)) || (n > 0) ) {

To get the desired condition, use
 if ( (m > 0) && ((n < -1) || (n > 1)) ) {
 if ( ((m < 0) || (n > 0)) && (n > m) ) {
 if ( ((m < 0) || (n < 0)) && ((m > 0) || (n > 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;
}
```

```
if ( ((m < 0) && (n > 0))
    || ((n > 0) && (m < 0)) ) {
```

The conditions on both sides of the `||` are equivalent.

```
} 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;
```

If both m and n are 0, it will never reach the second conditional, since it satisfies the condition before it.

[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;
  }
}
```

Yes for the first.

No for the second, since the only way to call verbose_equality on y and z is if z=x and z=y, in which case the first verbose_equality of x and y would have had output.

[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

}
```

```
void print_rational( int num, int den ) {
  std::cout << num << "/" << den << " = ";

  if ( num == 0 ) {
    std::cout << "0"
              << std::endl;
  } else if ( num%den == 0 ) {
    std::cout << (num/den) << std::endl;
  } else if ( num < 0 ) {
    if ( num/den != 0 ) {
      std::cout << (num/den) << " - ";
    } else {
      std::cout << "-";
    }

    std::cout << (-num%den) << "/"
              << den << std::endl;
  } else {
    if ( num/den != 0 ) {
      std::cout << (num/den) << " + ";
    }

    std::cout << (num%den) << "/"
              << den << std::endl;
  }

  return;
}
```

[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;
}
```

There are a number of poor coding practices including:

 - The first condition does not use parentheses, so it is more difficult to read the condition
   - It would be better if:
            if (a >= b && a >= c) {
     were written as
            if ((a >= b) && (a >= c)) {

 - Using the same variable name declared inside a body eclipses the variable with the same name outside that
   body. Thus, both local variables technically work, but its confusing.

   - It is a temporary varaible meant to
     temporarily store a value, so, give it a
     reasonable name. Contrast the following with
     the code above:

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

There is one major weakness, and it is a common weakness programmers make: that of performing unnecessary
operations.

Consider the case where a == b == 3 and c == 5

```
        if ( (a >= b) && (a >= c) ) {
          //Swap 'a' and 'c'
```

 - In this case, the condition is true, and we are swapping 'a' and 'c', even though they are equal, so we're
   wasting CPU cycles.

    - Thus, we should be using:

```
        if ( (a >= b) && (a > c) ) {
          //Swap 'a' and 'c'
```

  There is a bug, the second swap function does not work. This can be seen with the renamed temporary variable

```
      int tmp{ b };
      c = tmp;
      b = c;
```

[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;
}
```

```
int main() {
  for ( int i{ 0 }; i <= 9; i++ ) {
    std::cout << i << " ";
  }

  std::cout << std::endl;

  for ( int j{ 2 }; j <= 7; j++ ) {
    std::cout << j << " ";
  }

  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

}
```

```
void print_up_to( int m, int n ) {
  if ( m > n ) {
    return;
  }

  for ( int k{ m }; k <= n; ++k ) {
    std::cout << k << " ";
  }

  std::cout << std::endl;

  return;
}
```

[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

}
```

```
void print_up_to( int m, int n ) {
  if ( m <= n ) {
    for ( int k{ m }; k <= n; ++k ) {
      std::cout << k << " ";
    }
  } else {
    for ( int k{ n }; k <= m; ++k ) {
      std::cout << k << " ";
    }
  }

  std::cout << std::endl;

  return;
}
```

[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

}
```

```
void print_from_m_to_n( int m, int n ) {
  if ( m <= n ) {
    for ( int k{ m }; k <= n; ++k ) {
      std::cout << k << " ";
    }
  } else {
    for ( int k{ m }; k >= n; --k ) {
      std::cout << k << " ";
    }
  }

  std::cout << std::endl;

  return;
}
```

[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

}
```

```
void print_by_twos( int minimum, int maximum ) {
  if ( minimum > maximum ) {
    // Swap the two values
    int tmp{ minimum };
    minimum = maximum;
    maximum = tmp;
  }

  for ( int k{ minimum }; k <= maximum; k += 2 ) {
    std::cout << k << " ";
  }

  std::cout << std::endl;

  for ( int k{ minimum }; k <= maximum; ++k ) {
    if ( k%2 == 0 ) {
      std::cout << k << " ";
    }
  }

  std::cout << std::endl;

  return;
}
```

[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

}
```

```
double calculate_exponent( double x, int n ) {
  if ( n < 0 ) {
    x = 1.0/x;
    n = -n;
  }

  // Now, 'n' is guaranteed to be positive

  double result{ 1.0 };

  for ( int k{ 1 }; k <= n; ++k ) {
    result *= x;
  }

  return result;
}
```

[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

}
```

```
unsigned int factorial( unsigned int n ) {
  unsigned int factorial{ 1 };

  for ( int k{1}; k <= n; ++k ) {
    factorial *= k;
  }

  return factorial;
}
```

[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

}
```

```
int double_factorial( int n ) {
  if ( n < 0 ) {
    return 0;
  }

  int double_factorial{ 1 };

  for ( int k{n}; k > 1; k -= 2 ) {
    double_factorial *= k;
  }

  return double_factorial;
}
```

[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

}
```

```
int range_sum( int m, int n ) {
  if ( m > n ) {
    int tmp{ m };
    m = n;
    n = tmp;
  }

  int sum{ 0 };

  for ( int k{ m }; k <= n; ++k ) {
    sum += k;
  }

  return sum;
}
```

[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

}
```

```
unsigned int gcd( unsigned int m, unsigned int n ) {
  if ( (m == 0) || (n == 0) ) {
    return m + n;
  }

  int smaller{ m };

  if ( n < smaller ) {
    smaller = n;
  }

  int gcd{ 1 };

  for ( int k{ 2 }; k <= smaller; ++k ) {
    if ( (m%k == 0) && (n%k == 0) ) {
      gcd = k;
    }
  }

  return gcd
}
```

[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

}
```

```
unsigned int count_ones( unsigned long n ) {
  unsigned int count{ 0 };

  for ( unsigned long k{1}; k != 0; k <<= 1 ) {
    if ( k & n ){
      ++count;
    }
  }

  return count;
}
```

[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 );
```

```
unsigned int longest_run_of_ones( unsigned long n ) {
  unsigned int result{ 0 };
  unsigned count{ 0 };

  for ( unsigned long k{1}; k != 0; k <<= 1
  ) {
    if ( n & k ) {
      ++count;
      result = max( result, count );
    } else {
      count = 0;
    }
  }

  return result;
}

unsigned int longest_run_of_zeros( unsigned long n ) {
  return longest_run_of_ones( ~n );
}

unsigned int longest_run( unsigned long n ) {
  return max(
    longest_run_of_ones(  n ),
    longest_run_of_ones( ~n )
  );
}

unsigned int max( unsigned int m, unsigned int n ){
  if ( m >= n ) {
    return m;
  } else {
    return 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

}
```

```
bool is_power_of_two( unsigned long n ) {
  return (n != 0) && (n & (n - 1)) == 0;
}
```

[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

}
```

```
bool is_in_array( int n, int array[], std::size_t capacity ) {
  for ( std::size_t k{ 0 }; k < capacity; ++k ) {
    if ( array[k] == n ) {
      return true;
    }
  }

  return false;
}
```

[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

}
```

```
std::size_t first_in_array( int array1[], std::size_t capacity1, int array2[], std::size_t capacity2 ) {
  // Go through 'array2' up until it is no longer possible
  // to fit the entries in 'array1'.
  for ( std::size_t k2{ 0 }; k2 < capacity2 - capacity1 + 1; ++k2 ) {
    // Assume we have a match until we determine otherwise
    bool found{ true };

    // Go through 'array1' and if we ever find that two entries
    // are not equal, set 'found' to false and break.
    for ( std::size_t k1{ 0 }; k1 < capacity1; ++k1 ) {
      if ( array2[k2 + k1] != array1[k1] ) {
        found = false;
        break;
      }
    }

    // If we have gone through all of 'array1' and 'found' is still
    // set to 'true', we have a match: return the index of the
    // first entry.
    if ( found ) {
      return k2;
    }
  }

  return capacity2;
}
```